Deep Learning with State Spaces for Efficient Sequence Modeling

A Master's Thesis

Karl Ulbaek - s183931

July 12, 2024

Deep Learning with State Spaces for Efficient Sequence Modeling

Master Thesis
June 15, 2024

By
Karl Ulbæk

Published by: DTU, Department of Civil Engineering, Richard Petersens Plads, Build. 324, 2800 Kgs. Lyngby Denmark
www.compute.dtu.dk
ISSN: [0000-0000] (electronic version)
ISBN: [000-00-0000-000-0] (electronic version)

ISSN: [0000-0000] (printed version)
ISBN: [000-00-0000-000-0] (printed version)

Approval

This thesis has been prepared over six months at the Department of Applied Mathematics and Computer Science, at the Technical University of Denmark, DTU, in partial fulfilment for the degree Master of Science in Engineering, MSc Eng.

It is assumed that the reader has a basic knowledge in the areas of machine learning and deep learning.

Karl Ulbæk - s183931

Signature

15/06/24

Date

Abstract

Many modern deep learning tasks either are, or can be, framed as a sequence modeling problem. Since their introduction, the transformer has become the universally preferred deep sequence model across a wide range of modalities. However, the self-attention module, which enables the impressive modeling capabilities of the transformer, scales quadratically in sequence length, making transformers impractical or even infeasible for certain tasks. Recently, a new class of deep sequence models have emerged, which possesses favorable scaling in sequence length, and while also exhibiting expressive modeling capabilities. These novel models are termed deep state space models, and their core module termed the state space primitive (SSP). The fundamental idea underlying the SSP is to simulate the discrete sequence as a latent continuous-time state space of the form: $\dot{s} = As(t) + Bx(t)$, $y = Cx(t) + Dx(t)$. Compatibility between the discrete data and the continuous-time state space is established through an intricate learnable discretization procedure. This work thoroughly examines the fundamental theoretical ideas of the SSP, interprets its properties and draws parallels to other deep sequence models. Subsequently, it is showcased how the theoretical ideas are translated into practical SSPs, which may be incorporated into a deep sequence model. The deep state space models processes a sequence in directional left-to-right manner, which raises concerns in regard to their applicability in certain domains. Their inherent directional nature serves as the overarching subject for the experimental part of this thesis. Motivated by their importance in the context of self-attention, it was thoroughly investigated whether rotational positional embeddings could improve the deep state space model. The empirical results suggest that they do no benefit from the additional positional information. Subsequently the effect of making the models bidirectional was investigated. In this regard, the empirical results indicate that bidirectionality slightly improves performance but induces a substantial computational overhead and should be used with caution. Additionally, the results suggest that the models have impressive memory properties and generally do not suffer from a limited memory horizon. Finally, this work proposes to leverage the potency of directional pretraining, but subsequently at downstream finetuning morph the model bidirectional. This approach performs surprisingly well while requiring half the wall clock time to pretrain.

Post Hand-in Corrections

Clarifications

- Page 8 line 2 from the bottom: It is vaguely stated in the thesis thus it is now emphasis that all vectors are column-vectors but x_k in the context of X are row-vectors: $X = [x_0, x_1, ..., x_k, ..., x_{L-1}] \in \mathbb{R}^{L \times d} \rightarrow X = [x_0^\top, x_1^\top, ..., x_k^\top, ..., x_{L-1}^\top] \in \mathbb{R}^{L \times d}$, thus linear projections operate as: Wx_k and XW respectively, which is also how it is consistently done through out the thesis.
- Page 22 equation 33: when ZOH is derived $\bar{B}$ emerges as: $\bar{B} = (A)^{-1}(\exp(A\Delta) - I)B$ however when ZOH is later used in equation 67 page 36 it appears on the form $\bar{B} = (A\Delta)^{-1}(\exp(A\Delta) - I)B\Delta$. It is noted that these 2 formulations are identical.
- Page 30 line 23: To clarify: Each of the d token embedding features are processed individually by and independent SSP. In other works there are d token embedding features and d independent SSPs; One for each of the token embedding features.
- Page 46 line 16: "the dense linear layer" $\rightarrow$ "the dense state transition matrix".
- Line 72 line 17: Delete the following sentence since it it mentioned in the next bullet point: "By controlling for compute by employing a larger model which matches the wall-clock time of the bidirectional model."

Equation Corrections

- Page 21 equation 28: Bx_k should to the left of the integral and not the right. In equation 33 it gets moved to the correct side.
- Page 23 line 8 from the bottom: $(\bar{A}, \bar{B}, C, \Delta) \rightarrow (A, B, C, \Delta)$.
- Page 31 equation 54. $\bar{A}$ should be indexed in terms of k and not l. $\bar{A}^l \rightarrow \bar{A}^k$.
- Page 45 equation 75: Index off by one: $c_{k+1} = \left(1 - \frac{A}{k}\right) c_k + \frac{1}{k} B f_k \rightarrow c_k = \left(1 - \frac{A}{k}\right) c_{k-1} + \frac{1}{k} B f_k$.

Figure and Table Corrections

- Page 23 figure 4 caption: "source" is a clickable hyperlink when viewed in a PDF but it is colored black and it may easily be overlooked that it is in fact a clickable link.
- Page 48 line 3 and 7 from the bottom: "SSPs in deep sequences models" should be number 2 and "Introducing the Long Range Arena" should be number 1 in the listing.
- Page 55 figure 10: The classification task used in the right window is Cifar10-numerical and NOT Cifar10-categorical.
- Page 56 table 2: both in the table and in the table caption. PathX $\rightarrow$ Pathfinder.
- Page 66 table 5: 54 RoPE-configured models were trained not 64. 20 total non-RoPE models were trained and not 10 as it appears in the table. Evidently all the % are incorrect as well.

Equation/Figure/Table Reference Corrections

- Page 10 line 6 from the bottom: Figure 4.2.3 $\rightarrow$ Figure 1.
- Page 23 line 8: equation (61) $\rightarrow$ equation (36).
- Page 27 line 12: equation (61) $\rightarrow$ equation (36).
- Page 56 line 9: Table 2 $\rightarrow$ Table 3.
- Page 60 line 11 from the bottom: Appendix 17 $\rightarrow$ Appendix A.

- Page 64 line 8: ...blocks 9 right before... → ...blocks (see figure 9) right before...
- Page 66 line 12: "The employed RoPE implementation is a forked and modified version of this repository." *This repository* is a clickable hyperlink to the repository however it may easily be missed as it is colored black and needs to be hovered with the mouse in the PDF for the link to show.
- Page 69 line 11 from the bottom: Appendix C 17 → Appendix A.
- Page 74 line 7 from the bottom: Appendix C 17 → Appendix A.

Meaning Altering Spelling Mistakes

- Page 12 line 6: MLP → MLM.
- Page 17 line 24: "two. halves" → "second half".
- Page 32 line 18 from the bottom: "s4d: (Diagonal) Structured State Space Sequence Mode" → "s4d: (Diagonal) Structured State Space Sequence Model".
- Page 32 line 9 from the bottom: "s4: Structured State Space Sequence Mode" → "s4: Structured State Space Sequence Model".
- Page 36 line 1,2,3 from the bottom: data-dependent → input-dependent.
- Page 55 line 9: Test accuracy → Validation accuracy.
- Page 78 line 20-25: 1F → F1 and 1F1R → F1R1.

Glossary

All terms are explained along the way when they first introduced but for good measure an extensive glossary has been included. The terms appear in no particular order.

P:	The latent data generating process. A sample from is P is a sequence of tokens with some dependant and ordered relationship.
L:	The length of the sequence.
τ:	The token, the individual elements which constitutes the sequence $\mathcal{T}$. The tokens may either be considered categorical such as DNA strings or numerical such as stock prices, audio etc. No notational distinction is made between the categorical and continuous variants.
$\mathcal{T}$:	The sequence of L tokens sampled from P, such that $\mathcal{T} = [\tau_0, \tau_1, \ldots, \tau_k, \ldots, \tau_{L-1}]$.
`Tokenizer`:	The specifics of this operator depends on whether the data is considered categorical or numerical. In either instance the `Tokenizer` should be thought off as a preprocessor which cleans and standardizing the sequence. *Categorical*: The function splitting and mapping the raw sequence $\mathcal{T}$ into a one of the allowed categories. *Numerical*: When the raw data $\mathcal{T}$ is numerical valued (such as a times-series of floats) the data is not tokenized per say but simply normalized.
d:	The token embedding dimension. In the literature this quantity is sometimes also referred to as the channel dimension or feature dimension or model dimension or d_{model}. This work will try to be consistent and only use the term token embedding dimension.
x:	The token embedding which is a vector of dimension d. x is typically indexed using k such that x_k is the k'th token embedding in the sequence X. The entries of x are called token embedding features.
X:	The entire sequence of token embeddings of length L and width d. That is, $X = [x_0, x_1, \ldots, x_k, \ldots, x_{L-1}]$ and $X \in \mathbb{R}^{L \times d}$.
b:	The batch dimension. A batch of sequences X have shape $(b \times L \times d)$. b may also denoted the bias term associated with a linear projection matrix W.
f:	Some generic function.
y:	Used as a generic term to describe the output of some function f. Typically the function f operates on token embeddings x and y is just some transformed version of x with the same dimension.
t:	Used to index a continuous time variable or function.
T:	The total time given a continuous function or variable, for instance $y_t = f(t), \quad t \in 0 \ldots T$.
k:	Term used to index discrete variables.
Linear projection:	Matrix-vector-product possibly with a bias term b.
W:	A weight matrix used to perform a linear projection of usually the token embedding x. W is typically superscripted to display some form of association such as $W^{(Q)}$ to indicate that W is used to create the output Q. W typically has dimension $R \in d \times d$ and may implicitly include a bias term b.

σ:	Some arbitrary non-linear activation function. If not specified in the context then σ is typically the sigmoid activation function.
$\mathcal{K}$:	The continuous convolutional kernel.
$\bar{K}$:	The Discrete convolution kernel.
$*$:	Convolution operator.
$\oplus$:	Element-wise addition of two vectors or matrices of the same shape.
$\otimes$:	Element-wise multiplication of two vectors or matrices of the same shape.
$SISO$:	Single input single output. Used to describe a function/operator which takes a single scalar element (of a vector) and produces a single scalar output.
$MIMO$:	Multiple input multiple output. Used to describe a function/operator which takes as input multiple contiguous values of a vector and produces multiple contiguous output elements.
Maximal $MIMO$:	Used to describe a function/operator which takes a vector in its entirety and produces a vector of the same dimensions.
$\mathbb{A}$:	The attention matrix.
Temporal mixing:	This is also known as "time mixing". Temporal mixing implies mixing along the sequence dimension L, i.e. any transformation where embeddings from different tokens affect each other, that is mix.
Feature mixing:	This is also known as "channel mixing" and corresponds to mixing along the token embedding dimension d. This expression refers to any transformation which performs mixing but treats all token embeddings independently, that is mixing occurs only along the embedding dimension and no information flows between tokens.
Sequence primitive :	Any parameterized function/operator/primitive which performs *temporal mixing* on X . The sequence primitive may also entail *feature mixing*, but this is not required and often not the case.
Deep sequence model:	The entire deep neural network based on some sequence primitive. For instance the transformer is a deep sequence model based on the self-attention sequence primitive.
SSM:	State space model. Is in this work SSM refers exclusively to the classic state space model from equation (12).
SSP:	State space primitive. The term is used to denote any sequence primitive with theoretical roots in the state space theory.
s4	Is a specific SSP instance which uses the SISO formulation and a diagonal plus low rank decomposition of A.
s4d	Is a specific SSP instance which uses the SISO formulation and a diagonal A.
s6	Is a time variant SSP where the parameters (B, C, Δ) depends on the input thus s6 is linear time variant. s6 uses the SISO formulation and a diagonal. A.
Mamba (Mamba-block)	The network architecture/block used in the s6 paper.
Simple (Simple-block)	The network architecture/block used in the s4d paper.

s:	The state. The state is an N dimensional vector. Sometimes the state is also called the hidden state or latent state. All 3 terms are interchangeably but this work will try to be consistent and only use state.
G:	The input dimension of the SSP. For all practical purposes in this work $G = 1$, which implies a SISO formulation such that all features of the token embedding vector are processed independently.
$\mathcal{H}$	The number of heads (similar to multihead attention) in the SSP, that is $D/G = \mathcal{H}$. For all practical purposes in this work $\mathcal{H} = D$ due to the SISO formulation.
N:	The state dimension.
A:	The continuous state transition matrix, $A \in R^{(N \times N)}$.
B:	The continuous state input matrix, $B \in R^{(N \times G)}$.
C:	The continuous state output or emission matrix, $C \in R^{(G \times N)}$.
D:	The continuous state skip matrix, $G \in R^{(G \times G)}$.
Δ:	The discretization parameter, which is a scalar
$\bar{A}$:	The discretized state transition matrix, $A \in R^{(N \times N)}$.
$\bar{B}$:	The discretized state input matrix, $B \in R^{(N \times G)}$.
B_k, C_k, Δ_k	Are the input dependant variants used in s6.
$\bar{A}_k, \bar{B}_k$	Are the discretized input dependant variants used in s6.
$\mathcal{F}$	The Fourier transform.
Λ	A diagonal matrix.

Contents

1 Introduction

Sequence modeling is the core of modern deep learning as most modalities of interest are either sequences or may be formulated as sequences. The deep models used for sequence modeling are a versatile class of models able to operate on arbitrary inputs of sequences from a wide variety of domains such as natural language, audio, images, video, genomics and times series [79][29][24][72][86][39]. Since their introduction in 2017, the transformer[80] has become the predominant state of the art approach for deep sequence modeling. Self-attention is the underlying mechanism which enables the powerful sequence modeling capabilities of the transformer. The effectiveness of self-attention is attributed to its ability to densely map all possible paths of information exchange between all elements in the sequence. While this is what allows self-attention to model arbitrarily complex patterns in the data, it comes at the cost of quadratic scaling in the length of the sequence[57]. The quadratic scaling makes the transformer computationally impractical or even infeasible for modeling modalities which are inherently long and rely on long range dependencies such as DNA and audio. As a result, a large body of research has gone into combating the unfavorable scaling properties of self-attention and may be categorized into 2 general methodologies: 1) Sliding window attention[12], which simply limits how far forward and backward each element in the sequence is allowed to *attend*. Sliding window attention comes at the cost of the global receptive field of the model and thus its ability to model long range dependencies. 2) Linear attention[45], which implies removing a certain nonlinearity from the attention mechanism but comes at the of amputating self-attention of its core modeling properties which made it effective in the first place.

Prior to the transformer-era, deep sequences modeling was primarily undertaken using deep recurrent models such as recurrent neural networks[25] (RNN) and long term short term memory networks[41] (LSTM). Deep recurrent models exhibit favorable linear scaling in sequence length, making them applicable for long sequence modeling. Nonetheless, they possess two major shortcomings. The conventional recurrent models suffer from being difficult to train due to vanishing gradients[63]. Moreover, they are also slow to train, as their recurrent formulation disallows crucial parallelization along the sequence dimension. These two deficiencies make conventional recurrent architectures difficult to scale and undesirable in practice. On the contrary, scalability is where the transformer architecture excels[14], and is considered one of the primary reasons for its dominance of the deep sequence modeling landscape.

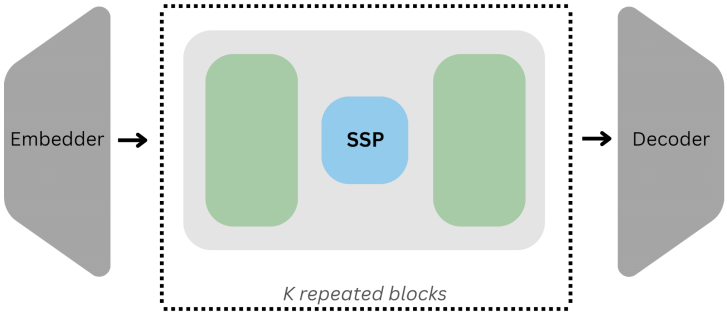

Figure 1: A **very generic** deep state space model. As other deep sequence models it consists of an embedding module, K repeated network blocks and a decoder module. At the core of each network blocks lies an SSP.

Recently however, the field of deep recurrent models have seen a sudden resurrection, followed by a surge of renewed interest. The resurrection was pioneered by a certain class of state space inspired deep models. These deep state space models can be thought of as being similar to the transformer but employing a **state space primitive** (SSP) in place of self-attention (see figure 1).

Conceptually the SSP implies modeling the *discrete* sequence using a *continuous* time state space[1] of the

form:

$$\begin{aligned} \dot{s}(t) &= As(t) + Bx(t) \\ y(t) &= Cs(t) + Dx(t) \end{aligned} \tag{1}$$

Where s is the state, x and y are the input and output of the state space. A is the state transition matrix, B is the input matrix, C is the emission matrix and D the skip matrix. The pivotal detail is the fact that the state space is continuous time, while the sequence is discrete. Evidently, the SSP simulates the discrete sequence as latent continuous time state space. Compatibility between the continuous time state space and the discrete sequence is facilitated by an intricate discretization procedure. From a purely mechanical standpoint, the discretization procedure is the primary quantity that distinguishes the deep state space models from other deep linear recurrent models.

The introduction of the deep state space model was quickly followed by a versatile fleet of approaches to modernizing the deep recurrent model[62][69][11][64]. The deep state space models and the modernized recurrent models comprise a worthy competitor to the transformer in regard to modeling expressivity and architecture scalability and matches or exceeds the performance of the transformer in a wide range of modalities and tasks[32][20][70][82][65]. Most importantly, the recurrent models (here under the deep state space model) remain linear or pseudo linear in sequence length, facilitating modeling tasks that involve sequences with millions of elements.

This thesis takes particular interest in the deep state space models, partly because they constitute a substantial milestone in the literature by demonstrating a 20% jump in average accuracy on a long range dependency benchmark suite[78]. Evidently, they exemplified the untapped potential of deep recurrent models and may be considered the instantiator of a renewed interest in deep recurrent models as a whole. Furthermore, deep state space based models have far reaching and profound theoretical ties to a wide variety of subjects. The core module of the deep state space model, that is the SSP, is heavily inspired by and builds upon the large foundation of the classical state space models. The classical models are typically associated with the Kalman filter, control theory and in general systems of ordinary differential equations (ODE). While closely related to, the SSP do not originate from the classical state space model, but rather from a perspective of optimal online function approximation theory (HiPPO-theory[33]), that highlights another theoretical connection. Even more interesting are the theoretical ties in regard to other deep learning models: For Instance the deep state space model poses a dual representation that allows it to either be computed as a recurrence one element at a time or as a global convolution over the whole sequence. Additionally, the discretization procedure may be perceived as a particular instance of learned gating mechanism analogously to the gating mechanism of a simple gated RNN[35]. While these theoretical perspectives are interesting in themselves, they also shed light on what enables successful deep sequence modeling and helped propagate the field of deep recurrent models.

Since the breakthrough of the first performant[34] SSP based deep sequence model, a wide variate of different SSPs have appeared. This thesis will take special interest in three particular SSPs. They are build on the same theoretical foundation but employ slightly different assumptions in regard to the state space equation (1). The motivation for each of the SSPs as well as how they differ is outlined briefly:

- s4: **S**tructured **S**tate **S**pace **S**equence Model[34]. s4 is considered the original breakthrough SSP, that brought attention to deep state space models and implicitly highlighted the potential of recurrent models in general. s4 is characterized mainly by imposing a diagonal plus low rank (DPLR) structure on the state transition matrix A.
- s4d: (**D**iagonal) **S**tructured **S**tate **S**pace **S**equence Model[36]. s4d is almost identical to s4, but rather than imposing DPLR structure on the transition matrix A, it simply imposes a diagonal structure. The idea of the diagonal structure was initially disregarded as it was theoretically considered less expressive. The significance and importance of s4d lies in the fact it drastically simplifies certain aspects of the SSP while in practice being (almost) equally performant compared to s4. As a result (almost) all subsequent deep recurrent (state space and non state space) models have adopted the diagonal state transition matrix A.

- s6: **S**elective **S**tructured **S**tate **S**pace **S**equence Model Computed with a **S**can[32]. s6 extends the state space equations to be time variant by making the parameters A, B, C depend on the current input x. This makes s6 *selective* in regard to what information it chooses to incorporate into its understanding of the sequence, in turn making it particular suited for tasks involving natural language. The fact that s6 is time variant violates certain properties and entails s6 to adopt the *scan* operation in order to remain computational feasible.

One immediate implication of employing deep state space models (and recurrent models in general) is the fact that they are inherently directional. That is, they process a sequence from left to right (or right to left), entirely analogous to how humans read a sentence. On the contrary, self-attention is inherently omnidirectional in the sense that it wires information directly between all pairwise combinations of elements in the sequence. For certain modalities, such as natural language, directionality appears intuitive, since it is how we process natural language ourselves. Furthermore, directional models are a candid choice for auto-regressive generation tasks, since they naturally align with next token prediction pretraining objective. However, some modalities are only directional by convention, such as DNA and certain tasks do not require a directional model, such as sequence classification. This naturally raises questions in regard to the applicability of directional models in these instances of non-directional data or tasks. These questions can not be addressed theoretically. Attempting to address questions related to directionality in the setting of deep state space models will serve as the primary experimental subject of this thesis.

2 Purpose of this Work

This thesis will provide a methodical and coherent exposition of the theory that underlies the SSPs. A principle and defining component of the SSP is the intricate discretization procedure. The purpose and implications of discretization will be interpreted and discussed. Finally the SSP will be related to mechanisms in other deep models such as RNNs and CNNs.

Once the SSP has been theoretically established, it is required to undergo a number of computational considerations and modifications before it may feasibly be integrated into a deep sequence model. These computational modifications will be elaborated in full and it will be showcased how different approaches amount to different variants of practical/concrete SSPs (s4, s4d, s6).

The final part of the thesis will be almost entirely experimental. All 3 SSPs (s4, s4d, s6) will be implemented and investigated to some extend. The overarching subject will be assessing the implications of the inherent directionality of the deep state space models. The experimental section is split into 4 parts, each investigating a different topic:

1. Establishing a baseline on a widely adopted long-range-dependencies benchmark suite. The baseline results serve to ensure that all models employed in this thesis perform similarly to what was reported in their respective papers. This will also allow for the assessment of different deep network architectures.
2. Investigating whether the addition of positional embeddings can improve classification accuracy of the deep state space models.
3. Investigating the potential increase in classification accuracy by incorporating the deep state space models with bidirectionality.
4. Investigate the effect of pretraining deep state space models before applying them to a downstream DNA classification task. In particular, it is of interest to investigate *alternative pretraining approaches* when the downstream task allow for bidirectional models.

It is noted that the theoretical section is directly related to deep state space models. On the contrary the subjects investigated in the experimental sections is somewhat agnostic to deep state space models. The experimental subjects are tied to the inherent directionality of the deep model and as such may generalize to any deep directional model and in particular to deep recurrent models.

2.1 Thesis Overview

This thesis is roughly composed of tree parts:

Theoretical: Section 4 introduces the deep sequence modeling framework in general terms and establishes relevant notation used throughout the thesis. Section 5 briefly highlights the shortcomings of self-attention and the conventional RNN. Section 6 derives the theoretical SSP in full and discusses its properties and relates it to other deep models.

Computational: Section 7 derives each of the 3 different practical SSP variants s4, s4d, and s6 and compares them. Section 8 showcases how s6 achieves computational feasibility, by demonstrating how a linear recurrence may be computed in parallel along the sequence dimension. Section 9 briefly establishes how the parameters of s4 s4d and s6 are initialized. Furthermore, section 9 touches upon recent empirical results, which suggest that SSP and HiPPO-theory might be unnecessary.

Experimental: Section 10 serves as a thorough introduction to the experimental work. Section 11 established the main dataset used to facilitate the experiments. Section 12 showcased the concrete deep sequence model architecture and lays out all the hyperparameters. Section 13 are the baseline results. Section 14 are experiments related to positional embeddings, 15 the ones related to bidirectionality and 16 the ones related to bidirectional pretraining.

3 Related Work

This section is not meant as a historic recap of the sequence modeling literature. Instead it focuses on modern derivatives of s4 and s4d as well as modern RNN based approaches to subquadratic sequence modeling.

3.1 State Space Theory Related Variants

- **[Gupta et al. 2022]** is the first example of a state space based approach that employs a diagonal transition matrix A rather than the much more involved diagonal plus low rank decomposition used by s4. Their approach compromises flexibility since 1: Their formulation requires the zero order hold discretization. 2: They compound the state space parameters B and C into a combined quantity. 3: They employ a row-softmax normalization of their Kernel to enforce stability.
- **[Smith et al. 2022]** is the first example of a maximal multiple-input-multiple-output (MIMO) state space formulation. Due to the MIMO formulation, the convolutional formulation becomes infeasible and they are the first to demonstrate that the state space recurrence can be computed efficiently using an associative parallel scan instead. As a result of their MIMO formulation, their state space primitive performs both temporal mixing as well as feature mixing. Consequently, they relax their use of dedicated feature mixing layers in their network architecture, such as GLU or MLP layers, which are otherwise required by single-input-single-output (SISO) formulations such as s4 and s4d. However, their MIMO formulations restrict their effective state size to be much smaller than that of s4/s4d, raising questions in regard to the memory capacity of their state.
- **[Hasani et al. 2022]** extends the state transition matrix A in the state space to take temporal correlation of incoming input samples into account. As a result, their method is the first instance of input dependent state transition but in a limited form which still allows for an efficient computation resembling the LTI convolutional formulation.
- **[Qin et al. 2023]**, **[Poli et al. 2023]** and **[Li et al. 2022]** all use global convolutions to model long sequences effectively and efficiently. Similar to the convolutional representation of s4/s4d, they employ a global, implicit kernel, that is a kernel parameterized by a fixed amount of parameters but which is adaptable to match any sequence length. How the three methods differ is in their approach to the implicit parameterization, which in the s4/s4d setting are the state space parameters (A, B, C, Δ). However, these three approaches do not have corresponding recurrent formulation and thus do not possess the ability to perform fast and efficient auto regressive generation.

3.2 Other Sub-quadratic Sequence Modeling Approaches

[Martin and Cundy 2017] is the first instance of a linear recurrent model computed in parallel along the sequence length using an associative scan. Evidently, they showcased the ability to efficiently compute a linear recurrent model. Although efficiency was demonstrated, their model was not particularly expressive. One of the first examples of an expressive linear RNN originates from the work of **[Orvieto et al. 2023]**. They propose the Linear Recurrent Unit (LRU) which is a simple linear recurrence with a diagonal hidden state transition matrix. In particular LRU emphasizes proper initialization of the recurrence parameters and normalisation of the recurrence computation. The LRU is covered in depth later in this thesis due to its ties to the state space based models.

Gating and Input Dependence. **[De et al. 2024]** is a recent follow up work to LRU that proposes the Real Gated Linear Recurrent Unit (RG-LRU), thus extending the LRU with a more sophisticated recurrence formulation. In particular, RG-LRU incorporates an input dependent gating mechanism similar to the selectivity mechanism of s6. In line with s6, they too find that an input dependent parametrization of the recurrence is crucial for natural language modeling tasks. Additionally, they highlight the role of the forget gate, including the ability to reset the state.

[Qin et al. 2023] proposes a novel approach to the forget gates on an architectural level. They introduce a learnable parameter, parameterized in a manner where it monotonically increases for each layer in the

network. As a result, each subsequent recurrent layer in the network forgets at a progressively slower rate. This introduces an inductive bias where earlier layers forgets quickly and thus emphasizes local dependencies. On the contrary, later layers forget more slowly and thus emphasizess global or long range dependencies.

Matrix Valued States. **[Qin et al. 2024]**, **[Beck et al. 2024]** and **[Peng et al. 2024]** are all instances of approaches where the hidden state is *matrix-valued*, that is of size $d \times d$ rather than $d \times n$ as in the instance of s4/s4d and s6. d is the token embedding dimension and n is a fixed value, thus the matrix valued states are typically much larger. The matrix valued states are based on a concept known as Bidirectional Associative Memories (BAM)[2], that uses a covariance update rule. In essence, the covariance update rule is simply an outer product between 2 different projections (keys, values) of the same token embedding. The matrix valued state then emerges as the element wise cumulative sum of all these outer products (typically subject to some forgetting or decay mechanism).

Linear Attention. The concept of linear attention and attention free models are a whole body of literature in itself and will not be covered on a model individual level. However, the following general concepts are highlighted: The essence of linear attention is the removal of the softmax operation from the self-attention mechanism[45], which allows for a recurrent or block recurrent formulation. This, among other things, implies subquadratic training complexity, linear memory usage and linear auto-regressive complexity. Linear attention comes at the cost of expressivity and has until recently[77] [85] been vastly inferior to regular self-attention. It should be noted that there is a strong resemblance between matrix valued recurrent models and Linear attention based models. For instance, the matrix valued RNN from **[Qin et al. 2024]** is derived from the perspective of RNNs but considers itself equivalent to the Gated Linear Attention approach proposed by **[Yang et al. 2023]**.

Hybrid Layers and Hybrid Architectures. Infini-attention proposed in **[Munkhdalai et al. 2024]** is a layer level hybrid approach between regular self-attention and a matrix valued state representation. Infini-attention splits the sequence into blocks that are each processed serially using self-attention, while also building up a matrix valued BAM. The BAM is not reset between blocks but instead continued and thus represents a compressed history of the whole sequence. At each block segment, the self-attention module is allowed to attend to the BAM representation as well as the segment itself. Evidently infini-attention has linear computational complexity and bounded memory complexity in the sequence length, while having a full receptive field at each layer compared to regular block or sliding window attention.

Both the s6 as well as the LG-LRU paper demonstrates a similar approach, however on an architectural level instead. That is, they interleave s6 or RG-LRU blocks with sliding window attention blocks, thus their overall architecture maintains linear complexity in sequence length. In both instances, their hybrid architecture outperforms the pure s6, pure RG-LRU and pure-self- attention(non-sliding window) architectures.

4 The Deep Sequence Modeling Framework

In order to establish how to construct a deep sequence model around an SSP, the deep sequence modeling framework must be formalized in general terms. Formalizing the general deep sequence model allows for the opportunity to showcase precisely where SSPs will be used in a deep sequence model and what role they play. How SSPs are derived theoretically and how they work mechanically are left for later sections.

This section establishes the concept of sequence data and highlights the downstream implications of numerical versus categorical sequence data modalities. Subsequently, the deep sequence model is decomposed into smaller building blocks and the concept of the sequence primitive is introduced. The sequence primitive is introduced to disentangle the generic modules of the deep model from those (the sequence primitives) that actually give the deep model character. Furthermore, the two most commonly used unsupervised pretraining approaches will be introduced. This is partly due to the fact that pretraining plays a crucial role in large scale sequence modeling in general but also because different approaches to pretraining will be investigated in the experimental section of this thesis. Finally, it is briefly explained how non sequential data may be treated as a sequence to be modeled by a sequence model.

4.1 Sequence data

The concept of sequence data departs from the notion that the data is i.i.d generated according to some latent random distribution. Instead, it is assumed that the latent generator is a random process denoted P. The random process P does not generate standalone independent observations but rather a sequence of observations $\mathcal{T} = [\tau_0, \tau_1, ..., \tau_k, ..., \tau_{L-1}]$ with some ordered and dependent relationship. The elements τ_k of the sequence are denoted tokens. Each sequence from the latent generating process corresponds to a sample from the process and each sample is still considered independent. Examples of sequence data include:

- Time series: Stocks, medical readings such as EKG, meteorological measures such as temperature, wind, pressure etc. In general, any sensory data measured repeatedly in time, where each subsequent measure in time is a token.
- Waves: E.g. speech or music.
- Trajectories: GPS trajectories and spatial trajectories in general. In this case, the spatial coordinate readings through time corresponds to the tokens.
- Video: Where each frame of the video corresponds to a token in the sequence.
- Clickstream data: That is, the sequence of actions through time as a user interacts with a website or application.
- Decision/Action trajectories from reinforcement learning or robotic tasks: Each subsequent action is considered a token.
- DNA or generally protein/molekyle chains: Where each subsequent amino acid or molecule in the chain corresponds to a token.
- Natural language: Each word may considered the token or each character may be considered a token. The sequence may be anything from a sentence to a whole book.

4.1.1 Numerical versus Categorical

Sequence data will be roughly separated into two overall variants, numerical and categorical.

Numerical sequences. Numerical sequences are numerical valued and typically represented as floats. Among the above examples, time series, waves, GPS trajectories, videos, and click stream data are considered numerical sequences. The numerical sequence is associated with an underlying latent generation process, P, describing a numeric value that evolves continuously in time. Numerical sequences thus involves sampling the underlying generation process. For instance, how the temperature is continuously changing throughout

the day but may only be measured/sampled every hour by the meteorologist. While the underlying process may be continuous in theory, the signal emitted by the process is in practice sampled discretely, thus being numerical measurements through time. Moreover, sampling the process necessitates a set sampling rate, which introduces a trade-off between precision and storage. Evidently, the sampling rate may entail severe dependencies when modeling the sequence. For instance, a model may become dependent on the sampling rate and generalize poorly to an identical signal sampled at a different rate.

Categorical sequences. DNA and natural language are instances of categorical sequences. The tokens from a categorical sequences are typically non numeric. The tokens are instead one of the finite allowable categories, for instance the different possible amino acids, the different characters in the alphabet, or different words in a vocabulary. Categorical sequences are considered independent of time and do suffer from the challenges associated with sampling rates.

Notice that **no** notational distinction will made with respect to the numerical versus categorical: In either setting the underlying generation process is denoted P from which a sequence of tokens $\mathcal{T}$ are sampled. Moreover, the commonly used terminology of continuous sequences versus discrete sequences has been deliberately avoided in favour of numerical versus categorical since discrete versus continuous serve a very particular meaning in the state space theory introduced in the next section.

Inherent directionallity. Often, sequences possess an inherent directional causality, the obvious instance being time series where the present cannot depend on the future as it is yet to happen. While time series may be inherently causal, it does not mean that the future is not correlated with the past or can not model the past. On the contrary some sequences are non causal and direction stems from convention rather than causal direction. i.e. DNA and to some extent language.

4.1.2 Tokenization

In a modeling setting, the "raw" tokens $\mathcal{T}$ are typically *tokenized*. The term tokenization has roots in the NLP (natural language processing) literature but will be used ambiguously. In the categorical setting, *tokenization* is the process of cleaning and assigning the tokens to one of the allowed categories. In the numerical setting, the "raw" tokens are real valued (floating point numbers) and *tokenization* topically implies a simple normalization procedure, such as rescaling the tokens to the interval $[-1, 1]$. In either instance *tokenization* amounts to a preprocessing procedure that cleans and normalizes the data.

The Token Embedding Procedure. After the data has been tokenized, the first step of any sequence model is to perform some extent of token embedding. Embedding a token is the procedure of exchanging the token with a d dimensional vector representation x_k. The embedding procedure is slightly different depending on whether the token is categorical or numerical.

- In the instance of a categorical token, the token is one-hot encoded and then multiplied with some embedding matrix of adequate dimension.
- In the instance of a numerical token **no** one-hot encoding occurs. Instead, the token is directly multiplied by some embedding matrix of appropriate dimension.

Notice that the inherently categorical token and inherently numerical token become indistinguishable after the embedding step. Moreover the theory and concepts introduced in this work exclusively applies to and operates on the embedding vectors, x_k, and makes no direct assumptions in regard to the origin of the token as either numerical or categorical.

The sequence of token embeddings. The entries in the token embedding vector, x_k, are called token embedding features. The sequence of L token embeddings is a matrix denoted X, such that $X = [x_0^\top, x_1^\top, ..., x_k^\top, ..., x_{L-1}^\top] \in \mathbb{R}^{L \times d}$. Notice that $x_k^\top$ are now row vectors and thus linear projections with X some weight matrix, W, always has W on the right side. Throughout the thesis, linear projection matrices

will typically have a superscript such as $W^{(s)}$. This superscript has no mathematical meaning but is merely used to distinguish between different weight matrices.

d is in the literature commonly denoted the "channel dimension" or "feature dimension" or "model dimension" or "d_model". While all these terms refer to the same quantity, an effort will be made refer to d only as the *token embedding dimension*.

4.1.3 Numerical Versus Categorical Considerations

Natural Language Processing (NLP) has emerged as the most active research area within sequence modeling. As a result, it has been at the forefront of advancements in the sequence modeling field. Thus, many popular concepts and methods have been developed under the NLP domain, therefore the categorical sequence domain, and may be unbecoming or non optimal in the numerical sequence setting.

Another thing to keep in mind is that tokens from numerical sequences typically have direct relative meaning: The temperature measurement of 2.1° is relatively closer to the measurement of 4.5° than it is to that of 10.3°. On the contrary, in the instance of a categorical sequence, the token "cat" is neither closer to nor further from "toilet" than it is "truck". Some relative meaning can be learned by the embedding layer or gained from using a pretrained embedding layers such as word2vec[56]. Evidently, if a numerical sequence is treated as a categorical sequence, it is stripped of its relative meaning. This exact phenomenon will play a role in the experimental section of this thesis (see section 10).

4.2 The Deep Sequence Model

A deep sequence model is a deep neural network to be used for sequence modeling. The main defining property of the deep sequence model is the sequence primitive it utilizes.

4.2.1 The Sequence Primitive

This section introduces and formalizes the term denoted the *sequence primitive*. The purpose of the term is to be able to precisely pinpoint a specific module in the deep sequence model. The sequence primitive is what gives the deep sequence model its unique character defining properties and what makes one deep sequence model distinguishable from another. Evidently, the term is used to disentangle a particular entity from the rest of the components of the deep model architecture. An obvious example is self-attention, which is the sequence primitive, while the transformer is deep model architecture. Other examples of sequence primitives include convolutions, recurrent layers and state spaces primitives.

Formally, the term *sequence primitive* will be defined as a generic function, v, parameterised by weights, θ, which maps a token embedding sequence, $X = [x_0, x_1, ..., x_k, ..., x_{L-1}]$, of arbitrary length, L, to a sequence, $X' = [x'_0, x'_1, ..., x'_k, ..., x'_{L-1}]$, of the same dimension:

$$X' = v_\theta(X), \quad \in R^{L \times d} \tag{2}$$

In words, the sequence primitive is a parameterized, dimension-preserving sequence-to-sequence transformation. The purpose of the sequence primitive is to perform *temporal mixing*, or more precisely; the sequence primitive is the module which is **responsible** for temporal mixing in the deep sequence model.

Temporal Mixing. Temporal mixing is any transformation that mixes the sequence along the sequence dimension L, i.e. any transformation where the different token embeddings interact and affect each other. The term temporal mixing will be adopted exclusively but may also be known as "time mixing".

Evidently what distinguishes one deep sequence model from another is the sequence primitive. What in turn characterizes the sequence primitive is how temporal mixing is performed. Notice that a sequence primitive may also entail *feature mixing*, but this is not required and often not the case.

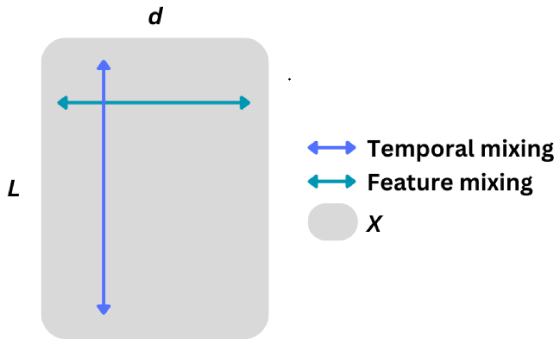

Figure 2: Illustration of temporal mixing compared to feature mixing on the token embedding matrix $X \in \mathbb{R}^{L \times d}$

Feature mixing. Feature mixing is any transformation along the token embedding dimension, d. That is, any transformation that performs mixing but treats all tokens independently, such that mixing occurs only along the embedding dimension and no information flows between tokens. The term feature mixing will be adopted exclusively but it shares meaning with the commonly used term "channel mixing". A generic formulation of the feature mixing would be:

$$X' = g_\theta(X) = [g(x_0), g(x_1), ..., g(x_k), ..., g(x_{L-1})], \quad \in R^{L \times d} \tag{3}$$

Notice that the formulation has g operate on the tokens embeddings independently. g could for instance correspond to a linear projection layer.

4.2.2 The Causal Sequence Primitive

Certain modeling tasks require a causal model. Causality implies that the output y_k is a function of at most the present x_k, and does not depend on information from future elements in the sequence $[x_{k+1}, ..., x_{L-1}]$.

Typically if the sequence primitive is causal, the deep model as a whole is causal. By definition recurrent models such RNNs and deep models based on SSPs are inherently causal. On the contrary, convolutional neural networks (CNN) and the transformer are not. The latter two may easily become causal by introducing some form of masking that turns their corresponding sequence primitives into masked self-attention and temporal/causal convolutions, respectively.

Causal primitives are often referred to as unidirectional as interaction or flow of information through the tokens happens in a one directional manner. On the other hand, non-causual models are considered bidirectional as tokens are allowed to affect each, both backward and forward in time. That is, information is allowed to flow in both directions of the sequence. The terms causal and unidirectional are considered equivalent and will be used interchangeably. The same goes for non-causal and bidirectional.

4.2.3 The Deep Sequence Model Architecture

An instance of a very generic deep sequence model architecture based on some SSP can be seen in 4.2.3. The deep sequence model architecture is typically composed of the following modules (see figure for generic visualizing):

An embedding module: $E_{\theta_{in}}(\mathcal{T}) : \mathbb{R}^{L \times in} \to \mathbb{R}^{L \times d}$**.** This layer operates on the tokens of the input sequence independently and servers to transform the input tokens into the desired embedding dimension d. See the above section on token embedding for additional details.

K neural network blocks: $F_{\theta_k}(X) : \mathbb{R}^{L\times d} \to \mathbb{R}^{L\times d}$**.** At the core of each block resides the sequence primitive that characterizes the deep model. Additionally, almost all modern blocks include one or more residual connections, normalization layers and nonlinearities. Other components frequently included in a block are linear projections layers, gating connections, multilayer perceptrons etc. The K blocks are typically identical in design and stacked one after another but each have their own set of independent weights θ_k.

A decoder module: $D_{\theta_{out}}(X) : \mathbb{R}^{L\times d} \to \mathbb{R}^{out}$**.** The shape and purpose of this layer is task dependent. For instance if the model is used for classification, R^{out} would correspond to the number of classes. Often this layer will not operate on the whole of $R^{L\times d}$. Intstead it would rely on some intermediate sequence-length-invariant pooling operation (`Mean` or `Max`), such that $\text{pool}(X) = \mathbb{R}^{L\times d} \to \mathbb{R}^{1\times d}$. The pooling operation may also be omitted entirely by introducing a classification token (`cls-token`) or by simply considering only the last token of the output sequence.

The full deep sequence models emerges by stacking the modules serially which can be expressed as a function composition:

$$\mathbb{F}_\Theta(\mathcal{T}) = \left\{D_{\theta_{\text{out}}} \circ F_{\theta_{K-1}} \circ \ldots \circ F_{\theta_1} \circ F_{\theta_0} \circ E_{\theta_{in}}\right\}(\mathcal{T}) = \tilde{\mathcal{T}}, \qquad \mathbb{R}^{L\times \text{in}} \to \mathbb{R}^{\text{out}} \tag{4}$$

Where $\circ$ is the function composition operator. Notice that the output $\tilde{\mathcal{T}}$ is task dependant and can take on a variety of different shapes. All models employed for classification in this work uses the `mean` pooling operation. In instances where the model is undergoing pretrained, no pooling operation is applied since the full sequence is required to compute the loss. Thus, the output will instead be of shape $R^{L\times c}$, where c is the number of to token categories.

Propagating the input sequence through the full model, that is computing $\mathbb{F}_\Theta(\mathcal{T})$, is referred to as computing the forward pass of the model. The parameters of the deep sequence models are optimized in the usual deep learning manner: computing the forward pass, computing a task dependant loss, back propagating the loss through the network (the backward pass) to compute the gradients of the parameters w.r.t. loss. Finally, the parameters are updated through some variant of gradient descent. Only the efficiency of the forward pass of the model is of interest in this thesis because it is assumed that the backward pass may be computed efficiently by some automatic differentiation framework.

4.2.4 The State Space Primitive (SSP)

The term *state space primitive* is used to denote any sequence primitive (intended for use in a deep seqeunce model), which is based on or relates to state space theory.[1] s4, s4d and s6, briefly introduced in the introduction, are all instances of concrete and practical state state primitives applicable in a deep sequence model.

4.2.5 The Deep State Space Model

The Deep State Space model is a deep sequence model that uses an SSP as its sequence primitive. This thesis will employ deep state space models based on the s4, s4d and s6 state space primitives. The phrase "SSP based models" or "SSP based deep models" are both used to refer to some collection of deep sequence sequence model that uses some variant of an SSP as their sequence primitive.

4.3 Unsupervised Pretraining in Sequence Modeling

The effectiveness and importance of pretraining in deep learning is well established, dating back all the way to the very start of the deep learning era [46]. Learning problems involving sequence data enables a particular kind of unsupervised pretraining. The following section on pretraining uses the work of **[Yang et al. 2019]** as source material.

[1] The terms state space primitive and sequence primitive as a whole are inventions of this thesis and generally do not appear in the literature. The adoption of the terms were deemed necessary as the literature is ambiguous and without consistent and precise terminology.

Sequence modeling pretraining can be split into two general methodologies and are most commonly referred to through the language models associated with them, that is, GPT-style[14] and BERT-style[22] pretraining. GPT-style pretraining is also known as autoregressive pretraning or next token prediction (NTP) pretraining. This thesis adopts the term next token prediction (NTP) pretraining. The BERT-style pretraining is sometimes refereed to as autoencoding pretraining or masked language modeling (MLM) pretraing. The latter more descriptive terminoligy of masked language modeling (MLP) pretraining is adapted. Both concepts will be elaborated below. The source material used for this subsection is the paper [87].

4.3.1 Next Token Prediction (NTP)

One key distinction is that NTP requires a uni-directional, casual model that is, while MLM may use a non causal model i.e. a bidirectional model. NTP employs the idea of modeling the joint distribution of the sequence as the factorized product of conditional probabilities.

$$p(\mathcal{T}) = \prod_{k=0}^{L-1} p(\tau_k \mid \tau_0, \ldots, \tau_{k-1}) \tag{5}$$

In words, given all the previous tokens in the sequence, what is the most likely next token. One immediate consequence of NTP pretraining is the fact that tokens are modeled as only depending on tokens backwards in sequences. This is a strong assumption that rarely holds. A more reasonable assumption would be that the probability distribution of token τ_k is accurately modeled using the whole sequence.

The strong appeal in NTP pretraining lies in the simple, yet potent loss signal it enables. All the model does is to causally predict the *next token* which it has not yet seen. There are $L-1$ *next tokens* in the sequence, implying that the model will perform a corresponding amount of predictions. In practice, calculating the loss reduces to calculating the loss of the final output of the model with respect to the original input shifted by one token.

$$\mathcal{L} = \frac{1}{L-1} \sum_{i=1}^{L-1} l(\tilde{\tau}_i, \tau_i) \tag{6}$$

(Notice i starts at 1 due to the shift.) Where $\tilde{\tau}_i$ is the predicted, adequately shifted token. l is typically the cross entropy loss for categorical tokens or the squared error loss in the instance of numerical tokens. Consequently, propagating one sequence through the network does not imply one classification task and a single unified loss but instead $L-1$ classifications and correspondingly many loss signals.

4.3.2 Masked Language Modeling (MLM)

On the contrary, the formulation of MLM is less grounded in probability theory. Instead, it involves simply masking out a percentage of the tokens (15% in the original BERT-paper) and then have the model predict the missing/masked tokens. However, when predicting the missing tokens, the model has access to the whole sequence, that is, both previous and future tokens, i.e. the model has bidirectional context of the missing tokens. As a result, MLM is considered more appropriate than NTP for sequence classification tasks such as sentiment analysis or DNA species classification.

From a theoretical standpoint, MLM ruins the models ability to model the sequence as a joint conditional probability, since the masked tokens ruins the probabilistic formulation given in equation (5). Evidently, MLM enforces the model with a notion of independence between the tokens rather than dependence as in the NTP setting.

From a practical standpoint, MLM introduces an artificial `[MASK]` token. Exposing the model to an artificial token, which does not appear in the true data distributing, may negatively distort the model's view of the data[87]. As a result, MLM may require more downstream fine-tuning to compensate the distortion caused by artificial the mask token.

Notice how the MLM loss objective only requires the model to predict 15% of the tokens in the sequence. On the contrary, NTP has the model predict ∼100% of the tokens in the sequence. In this sense, MLM amounts to $100\%/15\% = 6.67$ times less predictions than NTP and intuitively MLM should produce a relatively less potent loss signal.

The experimental section of this thesis will attempt to investigate the practical implications of either pretraining approach on a DNA classification task.

4.3.3 Auto Regressive Generation and Pretraining

Auto regressive generation or auto regressive inference is the process described as: given some input sequence $[\tau_0, \tau_1, \ldots, \tau_{L-1}]$ of length L, let the model predict the next token τ_L in the sequence. Subsequently, the predicted token is appended to the original sequence and the combined sequence $[\tau_0, \tau_1, \ldots, \tau_{L-1}, \tau_L]$ is fed to the model. This process is repeated in an iterative manner until a desired amount of tokens are obtained or untill the model predicts a special `[End-Of-Sequence-Token]`. Non-causal models may be used for auto-regressive generation, however the NTP pretraining of a causal model mimics the exact behavior of auto regressive generation during pretraining. As a result, NTP pretrained models will work out of the box with little to no downstream tuning for auto-regressive generation and are the preferred choice for this task.

4.3.4 Other Sequence Pretraining Approaches

NTP and and MLM are the most widespread pretraining methods, however alternatives and extensions have been proposed. The work of **[Yang et al. 2019]** proposes a novel hybrid pretraining approach that attempts to combine the advantages of NTP and MLM while avoiding their drawbacks. Their proposal is directly tied to self-attention and will thus not be discussed theoretically or attempted experimentally.

A recent work, **[Gloeckle et al. 2024]**, proposes an extension of NTP called multi token prediction. This extension has the causal model predict the next 2, 3 or 4 tokens. However, the benefit of their method seems to scale with model size and only outperforms NTP when applied to sufficiently large models (+500 million parameters), making it irrelevant in the setting of the experimental work conducted in this thesis.

4.4 Modeling Non-sequential Data Using a Sequential Model

Deep sequence models have expanded their scope beyond inherently sequential modalities, as first exemplified by the Vision Transformer [24]. The vision transformer processes images by partitioning them into patches and treating the patches as tokens in a sequence. Treating the image as a sequence corrupts the global spatial information of the image. To remedy the loss of global information, 2D positional encoding is added to the patches/tokens. While images are inherently non casual, so is the pure self-attention.

On the contrary, deep recurrent models, including deep state space models, are inherently causal and designed for genuine directional modalities. Nonetheless, deep state space models have been applied to non causal data and non sequential problems, such as image segmentation, classification and generation[52][89][84]. In all instances, the performance has been equivalent and often surpassing the state of the art, while providing linear scaling in image resolution. A common denominator for all approaches is to incorporate either bi or poly directionality to alleviate the inherent directional inductive bias of the state space mechanism. As mentioned, biases do not align with that of for instance images and would only impede the modeling capabilities in these instances.

This thesis will experimentally apply and access a similar methodology but in the sequential setting. That is, incorporating deep state space models with bidirectionality in relevant sequence tasks such as DNA or text-sentiment classification in the hope that it will alleviate the directional information gap and improve performance.

5 Sequence Modeling with Self-attention And Conventional RNNs

This section is considered supplementary material and will be a minor digression from the main topic of this thesis. The purpose of this section is to elaborate and support the claims made in the introductory parts in regard to the quadratic scaling of self attention and the linear scaling but also inherent shortcomings of the traditional RNNs.

5.1 Self-attention

The concept of self-attention was popularized (although not invented) in the 2017 paper "Attention is all you need" [80]. They dubbed their self attention mechanism the scaled dot-product attention. In its simplest form and decomposed into three equations for clarity, the scaled dot-product attention looks as follows:

$$\begin{aligned} &Q = XW^{(Q)}, \quad K = XW^{(K)}, \quad V = XW^{(V)} && \in \mathbb{R}^{L\times d} \\ &\mathbb{A} = \text{softmax}_{\text{row}}\left(\frac{QK^T}{\sqrt{\text{d}}}\right) && \in \mathbb{R}^{L\times L} \\ &Y = \mathbb{A}V && \in \mathbb{R}^{L\times d} \end{aligned} \tag{7}$$

- Q, K, V are called the queries, keys and values and emerge from three different linear projections of the input sequence, X, with the three distinct projection matrices $W^{(Q)}, W^{(K)}, W^{(V)}$. (Notice that the superscript of the projection matrix is an identifier and displays some form of relationship.)
- The attention matrix $\mathbb{A}$ is the outer-product between the queries and the keys, scaled according to d and softmax normalized along the rows.
- Y is the output sequence and is simply the matrix product between the attention matrix, $\mathbb{A}$, and the values, V.

The attention matrix, $\mathbb{A}$, is of special interest. At its core, the entries of $\mathbb{A}$ corresponds to the dot products between all pairwise combinations of token embedding vectors from X. Since there are L^2 unique pairs of token embeddings in X, computing the attention matrix requires computing L^2 dot-products. As a result, computing the attention matrix scales quadratically in L, and as a whole, self-attention scales quadratically in the sequence length.

A popular method for circumventing the quadratic scaling is to employ what is known as sliding window attention[12], also commonly referred to as local attention. In sliding window attention, each token is only allowed look (attend) forward or backward a certain amount of entries in the sequence. As a result, the attention matrix becomes a multi-diagonal where the width of the diagonal corresponds to the width of the sliding window (see figure 3). The computation associated with each token becomes bounded by the window size, and sliding window self-attention thus scales linearly in sequence length. Sliding window attention comes at the cost of expressively since each token has a limited local receptive field instead of a global receptive field. If sufficiently (depending on the sequence length) many sliding window attention layers are stacked, a global receptive may be achieved by the top layers (similar to CNNs). Evidently, sliding window removes the quadratic scaling but restricts or reduces the model's global or long range dependencies.

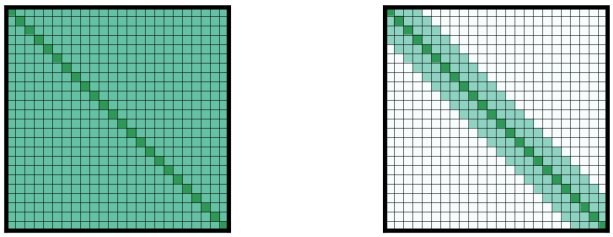

Figure 3: Left: The attention matrix of regular attention. Right: The attention matrix of sliding window attention with a window size of 7. Green entries imply that attention has been computed between two elements while white entries imply attention has not been computed.

For an in depth explanation of scaled dot-product attention and intuition related to the naming scheme of queries, keys and values, the reader is advised to consult the orignal paper [80].

5.2 RNNs

The concept of a recurrent neural network (RNN) dates back to at least 1990[25]. Today the term is used loosely to refer to any deep network that models the relationship between the tokens using some recurrent formulation. Traditional examples of RNNs are the simple recurrent network[25], Long short term memory network[41] and gated recurrent network[18]. Modern examples of recurrent models are the deep sequence models based on s4, s4d, s6 covered in this thesis but also the models discussed in the related work section 3.

A clear distinction will be made between the two generations of RNNs. The common denominator distinguishing the two variants are that the conventional are nonlinear, while the modern are linear in the recurrence, and each variant will often be referenced to as such. Even when distinguishing according to linearity, the term RNN is still ambiguous. For the sake of completeness, one variant will now be covered briefly. The RNN to be introduced is the conventional, nonlinear Elman network[25]: partly because of its rich history and important role in the development of the RNN field but also due to its simplicity. The Elman network is constituted by a number of subsequent layer. An Elman layer is given by the 2 equations:

$$\begin{aligned} s_k &= \sigma\left(W^{(s^1)}s_{k-1} + W^{(s^2)}x_k + b^{(s)}\right) \\ y_k &= \sigma\left(W^{(y)}s_k + b^{(y)}\right) \end{aligned} \tag{8}$$

(When ever the term RNN-recurrence is mentioned the reader should think of the above equation.) s_k is the (hidden) state and is a vector of dimension N. x_k is the k'th token embedding of the sequence X and is of dimension d. y_k is the k'th emitted token embedding of the recurrence and too has dimension d. $W^{(s^1)}, W^{(s^2)}, W^{(y)}$ and b^s, b^y are the linear projection matrices and corresponding bias terms of adequate dimensions. σ is a nonlinear activation function. On a general side note: The exclusion or inclusion of σ is what determines whether the RNN is linear or nonlinear. The first equation in (8) is the state equation and describes how the state is updated by combining information from the previous state s_{k-1} and information from the current token embedding x_k. The second equation in (8) describes how the output y_k is emitted from the current state s_k. The recurrent formulation implies that if one additional token embedding x_L is appended to the sequence, the corresponding emission y_L emerge by running the recurrence for a single additional step, i.e y_L has constant computational complexity $O(1)$, displaying that the recurrence as a whole scales linearly in L.

Moreover, this property displays why RNNs (conventional and modern) are preferable for the task of auto regressive generation. The most recent state contains a compressed representation of the whole sequence. Evidently, performing inference on a newly revealed token requires only the constant computation associated with propagating the final state one step and producing one emission. On the contrary, self-attention requires

the new token to be related to all previous tokens, implying O(L) computational cost per step (L steps then amount to O(L^2) complexity).

5.3 The Insufficiency of the Conventional RNNs

Despite their linear scaling properties, the conventional RNNs (not only the Elman network) has numerous crucial shortcomings making them undesirable in practice. Vanishing gradient problems[63] makes the conventional RNN difficult to train, in particular on longer sequences. The non linearity in the recurrent formulation prevents effective parallelization, causing inefficient hardware utilization and thus making it slow to train. Consequently, the conventional RNN is both difficult and slow to train, making it difficult to scale the architecture.

Meanwhile, parallelization, scalability and expressiveness are where the transformer and self-attention excels. These may be considered the defining characteristics that have led to the success and dominance of the transformer architecture.

6 Theoretical Derivation of the State Space Primitive

The purpose of this section is to show the theoretical derivation of the state space primitive (SSP) and subsequently discus its properties.

The theoretical foundation of the SSP originates from the work of four consecutive papers: **[Gu et al. 2020]**, **[Gu et al. 2021]**, **[Gu et al. 2021]** and **[Gu et al. 2022]**. Each paper builds on the previous with the two former primarily being theoretical and the two latter being more oriented towards computational feasibility and general applicability. The lead author behind the work and arguably the founding father of the SSP-theory is Albert Gu. While unknown to most, he deserves to be mentioned by name given the subject of this thesis. The following sections will follow the work of Albert Gu by introducing and formalizing the relevant theory and ideas that eventually amounts to the SSP. Unless otherwise stated, the source material for the following theoretical sections is the work of [35], but more specific references will be supplied in numerous instances. The overarching notion behind the work of Albert Gu can be expressed by the question: *Can we derive an expressive sequence primitive by simulating our sequence as a linear continuous time state space model?* The term state space model (SSM) will in this thesis exclusively refer to the SSM in its classical form as defined in equation (12). Linear continuous time SSM's are a subclass of the broader field of ordinary differential equations (ODE). Thus, the point of departure for deriving the SSP are ODE's.

6.1 An Ordinary Differential Equation Perspective

As a prerequisite, a generic formulation of an ODE will be introduced and discussed briefly. The introduction is based off if appendix C.1 from **[Gu et al. 2021]**. Consider the initial value problem ODE:

$$\dot{s}(t) = f(t, s(t)), \quad s(t_0) = s_0 \tag{9}$$

The unknown variable in the equation is the function $s(t)$. The equation is given in terms of change in the unknown function $s(t)$ over time, i.e. in terms of its derivative. The unknown that solves the equation is the function $s(t)$ that has the derivative $f(t, s(t))$. There are infinitely many functions that solve the equation. Unique solutions arise from conditioning the unknown function $s(t)$ on a predefined initial value at the initial time $t = t_0$. This is formalized by the two. halves of equation (9).

The formulation of the ODE (9) is known as its differential form and the immediate solution $s(t)$ emerges from integrating both sides:

$$s(t) = s_0 + \int_{t_0}^{t} f(z, s(z))dz \tag{10}$$

This is known as the integral form of the ODE. The integral form is a theoretical solution and almost entirely intractable in practice. However, by imposing sufficient structure on the dynamics of the system $f(t, s(t))$, for instance by assuming a linear continuous time SSM, a closed form analytical solution emerges. Moreover, the system dynamics $f(t, s(t))$ are in practice only available through discrete-time input observations x_{ti}. Consequently, $s(t)$ can only be estimated at discrete time instances t_i by iterating the equation:

$$s(t_{i+1}) = s(t_i) + \int_{t_i}^{t_{i+1}} f(z, s(z))dz \tag{11}$$

As a result of the discrete inputs the integral must be approximated. Different approximation schemes of the integral leads to different discretization techniques.

In conclusion, if some sequence of data is to be simulated as a continuous ODE, at least two subjects need be addressed.

- Sufficient structure (a SSM) must be imposed on the ODE such that the system has a feasible solution.

- Given the discrete nature of the sequence, some procedure (discretization) must be employed to establish compatibility between the discrete data and the continuous model.

Writing equation(11) in terms of an SSM as well as how to discretize accordingly will be elaborated on in subsequent sections.

6.2 The Classical State Space Model (SSM)

The SSM[1] is in its most generic form given by the two equations:

$$\begin{aligned} \dot{s}(t) &= A(t)s(t) + B(t)x(t) \\ y(t) &= C(t)s(t) + D(t)x(t) \end{aligned} \tag{12}$$

(The term state space model and its abbreviation SSM will be used exclusively to refer to the two equations given above.) The first equation represents the dynamics of an N'th order differential equation such as (9) rephrased as N first order equations. In words, it represents the state of the system, or more precisely, the change in state of the system. The second equation is the emission or observable of the first equation. As such, the SSM is a linear continuous time system of ordinary differential equations. The SSM defines a mapping from an input function, $x(t)$, through a state, $s(t)$, to an output function, $y(t)$. It is emphasised that the SSM is a continuous **function to function** mapping where all parameters and variables are continuous functions of time.

- $s(t)$ is the (hidden) state of size N.
- $x(t)$ is the input to the state and $y(t)$ is the output (emission) of the state and are both of size G.
- $A(t)$ is an $N \times N$ state transition matrix.
- $B(t)$ is the $N \times G$ input matrix.
- $C(t)$ is the $G \times N$ output (emission) matrix.
- $D(t)$ is an $G \times G$ feed through (skip) matrix.

Notice that the SSM is a function-to-function map of dimension $\mathbb{R}^G \to \mathbb{R}^G$. On the contrary, the generic sequence primitive from section 4.2.1 was defined as a sequence-to-sequence map of dimension $\mathbb{R}^d \to \mathbb{R}^d$. The SSP to be derived momentarily based on the above SSM formulation will similarly become a sequence-to-sequence map, however the input and output dimension of G will remain. The reason for the deliberate discrepancy between d and G will enable additional flexibility for the SSP and is explained in full in section 7.

6.2.1 Applications of the Classical State Space Model

The notion of a hidden state stems from applications (for instance the Kalman filter[7]) where the state itself is of interest, while being entirely unobservable. In these settings the state is available only through its (noisy) emissions $y(t)$ and the goal is to infer the state in an online manner as new observations become available.

In other applications, such as control, $x(t)$ is not considered observations but rather actions or control input. In these settings, the goal is typically to determine a sequence of input controls, $x(t)$, such that the system behaves in a desired way. In both Kalman filtering and control, (A, B, C, D), are often interpretable matrices originating from some physical understanding of the system, originally expressed as differential equations. However, the matrices may also be considered learnable parameters fitted according to the observed data. The SSP will be an instance of the learnable setting. Nonetheless, SSP is not deprived of meaningful interpretation as the parameters are often initialized based on optimal online function approximation theory briefly covered in 9.

6.2.2 Deriving the State Space Primitive

The derivation of the SSP is initialized by subjecting the SSM to two simplifications. The first simplification is a matter of notational convenience. That is, the $D(t)$ matrix will from here on out be excluded or considered 0, as it may in the context of deep network architectures be seen as a residual connection and thus have its functionally incorporated separately in the network block. The second modification is making the system dynamics invariant with respect to time. This is achieved by eliminating the parameters' dependence on the time variable, thus degrading the dynamics to a particularly well behaved linear time invariant (LTI) system[59]. Applying the two modifications materializes the SSP as a simplified LTI system.

$$\begin{aligned} \dot{s}(t) &= As(t) + Bx(t) \\ y(t) &= Cs(t) \end{aligned} \tag{13}$$

This formulation will be denoted the continuous SSP, or the continuous representation of the SSP. At this stage, the SSP is an LTI system and consequently exhibits two properties related to linearity and time invariance.

6.2.3 Linear Time Invariant System

First linearity is considered. Let the SSP be a parameterized operator, $y = SSP(A, B, C)(x)$, that performs the function mapping operation formulated by the above equations (13). It is a linear map in terms of input and output, implying:

$$\begin{gathered} \mathrm{SSP}(A, B, C)\,(x_1) + \mathrm{SSP}(A, B, C)\,(x_2) = \mathrm{SSP}(A, B, C)\,(x_1 + x_2) \\ \textit{and} \\ \mathrm{SSP}(A, B, C)\,(x_1 a) = \mathrm{SSP}(A, B, C)\,(x_1)\,a \end{gathered} \tag{14}$$

The 2. property is time invariance. Time invariance is enforced by making the parameters independent with respect to time and is mathematically formalized by the expression:

$$SSP(A, B, C)(x(t + T)) = y(t + T) \tag{15}$$

(Notice the above formula is subject to a minor abuse of notation since as of now $SSP(A, B, C)$ is defined as a function-to-function operator, while it in the above equation operates on function values.) In words, equation (15) states that an input *now* will amount to the same result (but shifted in time) as the same input at a *later* time.

Linear time invariant systems are a well studied and well behaved concept in classical signal processing and possesses a number of analytical properties. One property of interest prescribes that the mapping of the SSP (or any LTI system) may be recast as a continuous convolution[59]:

$$y(t) = (K * x)(t) = \int_0^\infty K(\tau)s(t - \tau)d\tau \tag{16}$$

$*$ is the convolution operator not to be confused with multiplication. $K(t)$ is the impulse response of the system and will emerge as the response/output of the system when the input is the Dirac delta function. Concretely, the SSP in its current form has impulse response:

$$K(t) = C\mathrm{e}^{tA}B \tag{17}$$

The significance of the convolutional formulation of the SSP will become apparent shortly.

6.3 Discretization

The SSP in its current form (13) is a continuous model that operates on functions, i.e. continuous signals. In section {4.1}, most sequence modalities were conceptualized as a product of an underlying process that generates a continuous signal $x(t)$, however it was also noted that in practice, this data is discretely sampled as $X = (x_0, x_1, ..., x_{L-1})$. Consequently, the SSP must be discretized to accommodate the concrete, discrete representation of the data. To achieve this, an additional parameter Δ is introduced. Δ can be interpreted in various ways, such as representing the resolution or time scale of the data. The discretization process using Δ assumes linearly spaced samples, such that $x_k = x(k\Delta), \quad k = [0, 1, ..., K-1]$.

Discretizing and reconstruction is a common task in signal processing and a variety of techniques exists, each with its own advantages and drawbacks. The simplest and most intuitive approach is Euler's method[6] defined as $x_k = x_{k-1} + f(x_{k-1})$. In practice however this approach is severely unstable for anything but short time horizons and small discretization values. Stable approaches exist and popular choices among these include *zero order hold* and *bilinear transform*, the latter of which is commonly known as the *trapezoid method*. The zero order hold method was specifically designed for the setting of discretizing LTI systems and will now be derived in full.

6.3.1 Deriving Zero Order Hold.

The source material for the derivation is [5] and [4]. Starting from the continuous SSP, and disregarding the emission/output equation:

$$\dot{s}(t) = As(t) + Bx(t) \tag{18}$$

Recall that discretizing corresponds to approximating the integral in the integral form of the differential equation (10). Thus, the first step is to convert the continuous SSP to integral form. The first step in doing so is to move $As(t)$ to the left hand side and subsequently premultiply both sides by e^{-At}:

$$\mathrm{e}^{-At}\dot{s}(t) - \mathrm{e}^{-At}As(t) = \mathrm{e}^{-At}Bx(t) \tag{19}$$

It is recognised that the left hand side corresponds to the derivative of the product-function $\mathrm{e}^{-At}s(t)$, which can be seen from $\frac{d}{dt}\left(\mathrm{e}^{-At}s(t)\right) = \mathrm{e}^{-At}\dot{s}(t) - \mathrm{e}^{-At}As(t)$. Substituting in the said product-function gives.

$$\frac{d}{dt}\left(\mathrm{e}^{-At}s(t)\right) = \mathrm{e}^{-At}Bx(t) \tag{20}$$

Multiplying through by $d\tau$ and subsequently integrating both sides in the interval 0 to t yields:

$$\int_0^t \frac{d}{dt}\left(\mathrm{e}^{-At}s(t)\right)d\tau = \int_0^t \mathrm{e}^{-A\tau}Bx(\tau)d\tau \tag{21}$$

The RHS integral is trivially solved which produces:

$$\mathrm{e}^{-At}s(t) - \mathrm{e}^{-A0}s(0) = \int_0^t \mathrm{e}^{-A\tau}Bx(\tau)d\tau \tag{22}$$

Then $\mathrm{e}^{-A0}s(0)$ is moved to the RHS and e^{-At} is multiplied through, corresponding to adding At to all exponents:

$$\mathrm{e}^{At-At}s(t) = \mathrm{e}^{At-A0}s(0) + \int_0^t \mathrm{e}^{At-A\tau}Bx(\tau)d\tau \tag{23}$$

A final reduction reveals the integral form of the continuous SSP.

$$s(t) = \mathrm{e}^{At}s(0) + \int_0^t \mathrm{e}^{A(t-\tau)}Bx(\tau)d\tau \tag{24}$$

Now, the actual discretization may commence. Let Δ denote the step size and let k denote number of steps such that the k'th state is defined as $s_k = s(k\Delta)$. Zero order hold applies the simple assumption that the input $x(t)$ is constant during each step, Δ. Plugging in $k\Delta$ yields:

$$s_k = \mathrm{e}^{Ak\Delta}s(0) + \int_0^{k\Delta} \mathrm{e}^{A(k\Delta-\tau)}Bx(\tau)d\tau \tag{25}$$

Similarly, incrementing the discretization step by 1 and plugging in $(k+1)\Delta$ gives:

$$s_{k+1} = \mathrm{e}^{A(k+1)\Delta}s(0) + \int_0^{(k+1)\Delta} \mathrm{e}^{A((k+1)\Delta-\tau)}Bx(\tau)d\tau \tag{26}$$

Rewriting the first term of the right-hand-side (RHS) as $\mathrm{e}^{A\Delta}\mathrm{e}^{A\Delta k}$ and splitting the integral into two parts at the boundary Δk gives:

$$s_{k+1} = \mathrm{e}^{A\Delta}\left[\mathrm{e}^{A\Delta K}s(0) + \int_0^{k\Delta} \mathrm{e}^{A(k\Delta-\tau)}Bx(\tau)d\tau\right] + \int_{k\Delta}^{(k+1)\Delta} \mathrm{e}^{A((k+1)\Delta-\tau)}Bx(\tau)d\tau \tag{27}$$

Recognize that what is contained in the square brackets is s_k from equation (25). Recall that the input, x, is assumed constant and thus moved outside the integral.

$$s_{k+1} = \mathrm{e}^{A\Delta}s_k + Bx_k \int_{k\Delta}^{(k+1)\Delta} \mathrm{e}^{A((k+1)\Delta-\tau)}d\tau \tag{28}$$

The next step is to get rid of the term $((k+1)\Delta-\tau)$, which is done my means of integration by substitution. Let the substitute function be $v(\tau) = ((k+1)\Delta - \tau)$. Evaluating the substitute function at the bounds of the integral.

$$\begin{aligned} v((k+1)\Delta) &= k\Delta + \Delta - (k+1)\cdot\Delta = 0 \\ v(k\Delta) &= k\Delta + \Delta - k\Delta = \Delta \end{aligned} \tag{29}$$

Additionally, notice that $\frac{dv(\tau)}{d\tau} = -1 \Rightarrow dv(\tau) = -d\tau$. Thus $dv(\tau)$ may be substituted for $-d\tau$ alongside new bounds defined from (29) which yields:

$$s_{k+1} = \mathrm{e}^{A\Delta}s_k - Bx_k \int_{\Delta}^{0} \mathrm{e}^{A(\tau)}d\tau \tag{30}$$

Switching the bounds of the integral reverses the sign such that:

$$s_{k+1} = \mathrm{e}^{A\Delta_k}s_k + Bx_k \int_0^{\Delta} \mathrm{e}^{A(\tau)}d\tau \tag{31}$$

Assuming A is non-singular, allows the use of $\int \mathrm{e}^{Az}dz = \left(\mathrm{e}^{Az} - I\right)A^{-1}$ which is simple integration of a matrix exponential and concludes the derivation of zero order hold by producing:

$$s_{k+1} = \mathrm{e}^{A\Delta}s_k + Bx_k\left(\mathrm{e}^{A\Delta} - I\right)A^{-1} \tag{32}$$

Gathering terms related to the state variable, s, and the input variable, x, yields the zero order hold discretization of the individual terms A and B:

$$\begin{aligned} \bar{A} &= \mathrm{e}^{\Delta A} \\ \bar{B} &= (A)^{-1}\left(\bar{A} - I\right) \cdot B \end{aligned} \tag{33}$$

The discrete recurrence of the state in terms of the discretized parameters $\bar{A}$ and $\bar{B}$ simply becomes:

$$s_k = \bar{A}s_{k-1} + \bar{B}x_k \tag{34}$$

(Notice the index has been shifted by 1 for cleaner notation compared to the derivation.)

The other stable discretization method is bilinear transform. Although it will not be derived, applying the bilinear transform to the continuous SSP produces the following discretized variants of A and B:

$$\begin{aligned} \bar{A} &= (I - 0.5\Delta A)^{-1}(I + 0.5\Delta A) \\ \bar{B} &= (I - 0.5\Delta A)^{-1}\Delta B \end{aligned} \tag{35}$$

Notice that A and B are the continuous variants of the SSP parameters, while $\bar{A}$ and $\bar{B}$ are used to emphasize that they have been discretized. In the context of discretizing the SSP, either method results in different parameterizations of $\bar{A}$ and $\bar{B}$. Theoretically zero order hold is an exact discretization of the system, whereas bilinear involves an approximation. Nonetheless, from a purely accuracy/expressiveness perspective, it matters little in practice which approach is used to the point where they may be used interchangeably. From a computational complexity stand point, it is observed that either approach relies on expensive matrix inversion, while zero order hold also requires an expensive matrix exponential. As such, bilinear appears the preferred choice. In subsequent sections of this thesis, it is revealed that structure (often diagonal) is imposed on A. Consequently, the computational expenses associated with matrix inversion and exponentiation becomes negligible and thus irrelevant.

6.4 The Recurrent Formulation of the SSP

The discretization process translates (A, B, C, Δ) into $(\bar{A}, \bar{B}, C)$. Applying the discretization to the SSP (13) converts the function-to-function representation to a sequence-to-sequence representation. The result is a recurrence in the state variable s:

$$\begin{aligned} s_k &= \bar{A}s_{k-1} + \bar{B}x_k \\ y_k &= Cs_k \end{aligned} \tag{36}$$

This formulation will be denoted the recurrent SSP or the SSP in recurrent mode/representation. The SSP in its recurrent formulation resembles that of the classical RNN. The primary and essential distinction being that the recurrent SSP is linear in the recurrence, whereas the traditional RNN typically has a nonlinearity in its recurrence as seen in equation (8). At first glance, omitting nonlinearities would intuitively imply a reduction in the model's expressiveness. An in depth discussion of the expressiveness of the SSP and further ties to the RNN can be found section 6.6. Moreover, the recurrent SSP sets itself apart from the RNN-recurrence through its incorporation of the discretization parameter Δ, resulting in a different parametrization of the matrices A and B. In the setting of a deep sequence model, Δ is a learnable parameter. Discretizing simply becomes the first step in the computational graph of the SSP and hence computed during the forward pass of the deep network and subsequently optimized accordingly.

6.5 The Convolutional Formulation of the SSP

In section {6.2.3} it was established that a continuous LTI system has a continuous convolutional formulation. An analogous property exists with regard to the discrete LTI systems. The discretization of the continuous SSP(13) had no impact on its LTI properties. Thus, the recurrent SSP has a corresponding discrete convolutional formulation.

Converting the analytical integral in equation (17) to an analogous sum is of no interest. For practical purposes a concrete formulation of the kernel is required. The kernel emerges from unrolling the recurrence in (61). Assuming the initial state s_{-1} is 0, unrolling the state recurrence by substituting, the expression for the state variable s from the previous iteration yields:

$$\begin{aligned} s_0 &= \bar{B}x_0 \\ s_1 &= \bar{A}s_0 + \bar{B}x_1 \\ s_2 &= \bar{A}^2\bar{B}x_0 + \bar{A}\bar{B}x_1 + \bar{B}x_2 \\ s_k &= \bar{A}^k\bar{B}x_0 + \bar{A}^{k-1}\bar{B}x_1 + \bar{A}^{k-2}\bar{B}x_2 + \ldots + \bar{B}x_k \end{aligned} \tag{37}$$

The k'th output, y_k, becomes.

$$y_k = Cs_k = C\bar{A}^k\bar{B}x_0 + C\bar{A}^{k-1}\bar{B}x_1 + C\bar{A}^{k-2}\bar{B}x_2 + \ldots + C\bar{B}x_k \tag{38}$$

Before proceeding, the attention is momentarily directed to the formulation of the discrete convolution: Given two sequences Z and V both of length L, the k'th value of the discrete convolution between the sequences is defined as.

$$(Z * V)[k] = \sum_{k=0}^{L-1} Z[L-k]V[k] \tag{39}$$

Let the k'th element of the sequence, Z, be expressed as $Z[k] = C\bar{A}^k\bar{B}$ and let V be the sequences of input data $V = X = (x_0, x_1, ..., x_{L-1})$. Performing these substitutions gives.

$$y_k = (Z * V)[k] = \sum_{k=0}^{L-1} C\bar{A}^{L-k}\bar{B}x_k \tag{40}$$

This expression is identical to (38), confirming that the unrolled recurrence is in fact a discrete convolution. Consequently, all outputs, Y, of the SSP can be obtained from performing one single convolution between the input sequence, X, and the particular kernel $\bar{K}$.

$$Y = \bar{K} * X, \qquad \bar{K} = \left(C\bar{B}, C\overline{AB}, C\bar{A}^2\bar{B}, \ldots, C\bar{A}^{L-1}\bar{B}\right) \tag{41}$$

This formulation will be denoted the convolutional SSP or the SSP in convolutional mode/representation.

It should be noted that the kernel is an implicit kernel as it depends on and is constructed from the separate parameters $(\bar{A}, \bar{B}, C, \Delta)$. Moreover, the kernel has the same length, L, as the input and is adapted dynamically if the length of the input were to change. Having the same length as the input further implies that the kernel is global and that all tokens have a global receptive field (still subject to the causality constraint). Additionally, notice that when the global convolutional is employed, all recurrent steps are performed in a single operation and the state itself is at no point materialised.

The significance of being able to express the SSP as a single convolutional operation lies in the fact that convolutions are easily and efficiently parallelizable, thereby allowing for fast and efficient training. Subse-

quently, at inference time, the SSP may be switched to recurrent mode to leverage the constant time and memory complexity of the recurrent formulation, thus achieving the benefits of both formulations.

6.5.1 Computation Cost of Convolutions

Addressing the computational complexity of convolutions may appear as a slight departure from the overall scope of this section. However, being able to compute convolutions efficiently is a crucial aspect and lies as a core building block for the development of the SSP theory as a whole. The source material for this section is the work of **[Selesnick and Burrus 1999]**.

The computational cost associated with the dense convolutional operation is the length of the kernel, M, times the length of the sequence, L. For this section, the length of the kernel is denoted M to highlight generality beyond the global convolution setting where M=L. The convolutional SSP given by (41) is an instance of a dense convolution, which due to its particular formulation implies a computational cost of $O(L \cdot M) = O(L^2)$, i.e. quadratic in sequence length. However, subquadratic approaches are available. One approach relies on the convolution theorem, that states that convolution in the time domain is equivalent to element wise multiplication in the frequency domain. Consequently, any convolution may be computed as.

$$Y = \mathcal{F}^{-1}(\mathcal{F}(\bar{K}) \otimes \mathcal{F}(X)) \tag{42}$$

Where $\mathcal{F}$ and $\mathcal{F}^{-1}$ are the fourier and inverse fourier transformers (FT) and $\otimes$ is element-wise multiplication. The fourier transform and inverse transform can be performed in O((M+L)log(M+L)) using the fast fourier transform (FFT) algorithm. Performing the convolution operation using this approach will be denoted the FFT convolution.

It is noted that frequency domain convolutions are not equivalent to dense convolution unless subject to specific precautions. Directly performing the operation in the frequency domain results in a circular convolution. Explaining circular convolution and how to enforce equivalency to dense convolutions are best done visually.

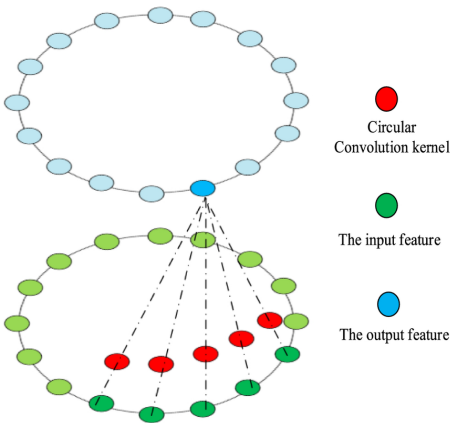

Figure 4: The circular convolution. The source of the image.

As indicated by the illustration in figure 4, the circular convolution treats the first and last token in the sequence as consecutive. Mathematical equivalence to the dense convolution is achievable through adequate zero padding of the input followed by adequate discarding of the output. The goal is to append or prepend a sufficient amount of zeros, such that the end of the input (when connected through the tiling of the kernel) is not able to exchange information with the beginning of the input. A concrete recipe that achieves the above goal is:

- Zero-pad the kernel with $L - 1$ elements.

- Zero-pad the input with $M-1$ elements.
- Perform the circular convolution, i.e the FFT convolution.
- Discard the first $M-1$ elements of the circular convolution.

The formula assumes $K \leq L$. The commutative property of convolution, $f * g = g * f$, allows the assumption to always be satisfied as the input sequence may simply be considered the kernel. The zero padding may be either post, pre or a combination as they amount to the same when the kernel/input is perceived as a circle.

Returning to the setting of the convolutional SSP. Recall that the length of the kernel is equal to the length of the sequence. Applying the FFT convolution first requires padding both the kernel and the sequence to double their lengths. Subsequently, two FFTs are required followed by element-wise multiplication and finally an inverse FFT. Evidently, although the FFT convolution remains log-linear in computational complexity, it does have a substantial constant term associated with it. While the FFT convolution provides log linear processing in sequences, the primary computational concern of the SSP lies not in the convolution operation itself but instead in efficiently generating the implicit kernel, $\bar{K}$. This subject will be addressed in later sections.

6.6 Interpreting the SSP

The previous sections provided the cornerstones of the state space primitive. This subsection will focus on discussing the SSP. Hereunder how to interpret the discretization parameter, and draw connections to RNNs and CNNs and lastly address concerns with regard to the expressiveness of the SSP due to its linearity.

6.6.1 Interpreting the Discretization Parameter Δ

(The following interpretation of Δ originates from section 2.4.1 of **[Gu 2023]**.) Aside from being entirely linear, the immediate distinguishing feature between the SSP and the classical RNN-recurrence is the discretization parameter Δ and hereunder the discretizated parameterization of $\bar{A}$ and $\bar{B}$. Nonetheless, some RNN's[71][81], have a similar interpretation as the SSP. That is, seen as an approximation to a continuous dynamical system. In this interpretation they too have an important reliance on discretizing the continuous system and in turn the discretization parameter Δ. However, these ODE interpreted RNNs view Δ as the step size of the data and considers it an inherent property of the data. Consequently, Δ is treated as a crucial tuneable hyperparameter. On the contrary, the SSP has Δ as a learnable parameter that represents timescale.

The time scale interpretation of Δ stems from the following observation. Consider an ODE of the form corresponding to the linear continuous SSM (13), $\dot{z} = cAz + cBx$. Additionally, consider the similar SSM $\dot{s} = As + Bx$ governed by the same system dynamics A and B. The former maps $x(c \cdot t) \to s(c \cdot t)$ if the latter maps $x(c) \to s(t)$. In this sense, they represent the same system but the former evolves at a rate c faster than the latter. The system dynamics persevere when subjected to scaling, as the scaling itself merely corresponds to changing the rate by which the system evolves.

Δ is the learnable parameter which becomes responsible for capturing and modulating the time scale of the data. If the system evolves rapidly, Δ should take on on a large value and vice versa. Concretely, Δ captures the length of dependencies proportional to $\frac{1}{\Delta}$. A rapidly evolving system implies large change and thus the dependencies should be short ranged. Evidently, when considering the SSP in its convolutional formulation, Δ may be interpreted as a quantity that allows for the length of the filters to be learned.

6.6.2 Discretization Corresponds to Learned Gating

This subsection will provide a parallel between the gating mechanism of some RNN and the discretization process of the SSP. The mathematical derivation originates from appendix C.2.2 in **[Gu et al. 2021]**.

Let the state of some RNN-recurrence be generically defined by the following gated recurrence.

$$s_{t+1} = \sigma\left(z_{t+1}\right) s_t + \left(1 - \sigma\left(z_{t+1}\right)\right) x_{t+1} \tag{43}$$

Where z is an arbitrary expression and σ is the sigmoid function. For the sake of completeness the term gating mechanism is elaborated: In the context of machine learning, the concept of a gating mechanism is loosely defined but typically involves two subsequent operations: An expression involving a nonlinearity followed by a point-wise multiplication with some variables or data. The nonlinear-expression is the gate, while the multiplication enforces which elements are allowed through the gate. Entries where the nonlinear expression evaluates to 0 (or close to 0) implies a closed gate, since the subsequent multiplication equates to zero. Thus, no data flows through the gate.

The gating in the recurrence, given by equation (43), is a *tied* gating mechanism: it first gates the hidden state, s, according to the gate, $\sigma(z)$, and subsequently gates the input, x, with the complimentary gate, $1-\sigma(z)$. The intuition being that, if x supplies relevant information which the state would want to incorporate, a corresponding magnitude of information needs to be erased from the hidden state to make room. Similarly, if no relevant is supplied by the current input, x, the gate is shut and the entire state, s, should be kept.

It is now shown how the gated recurrence 43 emerges as a particular discretization of a simple linear state space ODE. Let the simple 1D state space ODE have parameters $A = -1$and $B = 1$such that:

$$\dot{s}(t) = -s(t) + x(t) \tag{44}$$

The discretization choice that leads to the result is the backward Euler method given by:

$$w_{k+1} = w_k + \Delta_{k+1} f(k+1) \tag{45}$$

Inserting the ODE in the backward Euler yields:

$$s_{k+1} = s_k + \Delta_{k+1} \left[-s_{k+1} + x_{k+1}\right] \tag{46}$$

Rearranging by gathering Δs_{k+1} terms on the left hand side:

$$s_{k+1} \left(1 + \Delta_{k+1}\right) = s_k + \Delta_{k+1} x_{k+1} \tag{47}$$

Dividing through with $(1 + \Delta_k)$:

$$s_{k+1} = s_k \frac{1}{(1 + \Delta_{k+1})} + x_{k+1} \frac{\Delta_{k+1}}{(1 + \Delta_{k+1})} \tag{48}$$

Discretization demands a positive step size Δ. A simple way to ensure the arbitrary expression, z, in (43) to be strictly positive is using the exponential function and re-parameterize accordingly $\Delta_k = \mathrm{e}^{z_k}$. Inserting gives.

$$s_{k+1} = s_k \frac{1}{(1 + \mathrm{e}^{z_{k+1}})} + x_{k+1} \frac{\mathrm{e}^{z_{k+1}}}{(1 + \mathrm{e}^{z_{k+1}})} \tag{49}$$

Noting that a minor rearrangement of each of the terms on the RHS causes the sigmoid function to emerge. The 2 rearrangements are:

$$\frac{1}{1 + \mathrm{e}^{z_{k+1}}} = \left(1 - \frac{1}{1 + \mathrm{e}^{-z_{k+1}}} = \right) = 1 - \sigma\left(z_{k+1}\right), \qquad \frac{\mathrm{e}^{z_{k+1}}}{1 + \mathrm{e}^{z_{k+1}}} = \frac{1}{1 + \mathrm{e}^{-z_{k+1}}} = \sigma\left(z_{k+1}\right) \tag{50}$$

Substituting the two sigmoid expressions from above into (49) produces the desired gated recurrence.

$$s_{k+1} = \sigma\left(z_{k+1}\right) s_k + \left(1 - \sigma\left(z_{k+1}\right)\right) x_{k+1} \tag{51}$$

Deriving the gated recurrence through discretization of a state space ODE implies:

- The SSP and the RNN-recurrence both approximate the same underlying differential equation.
- The timescale Δ can be viewed as a learned gating mechanics.
- Lastly, the derivation supports the notation that the SSP (while being fully linear) does not have reduced expressiveness compared to traditional RNNs as one emerges as the special case of the other.

It should be noted that the above results are not without reservations. Although the derivation produced exact results and did not rely on approximations, the derivation requires a specific discretization scheme as well as a specific formulation of the gating mechanisms. Similar results may exist for other discretization schemes and gating mechanisms but has not been shown. Furthermore, the derivation makes one implicit assumption, which is incompatible with certain SSP formulations. In the derivation, the gating mechanism, z, is defined to be an arbitrary expression. For a range of concrete RNN formulations, z is dependent on the input data, x. Consequently z is time varying, implying that Δ (being a parametrization of z) is also time varying. The SSP recurrence 61 can be reformulated to account for time variance/dependence, however the convolutional formulation 41 requires a time invariant system and thus becomes infeasible. Later sections of this thesis will introduce one such time variant SSP (s6) and showcase alternative hardware efficient approaches to compute the recurrence.

6.6.3 Convolutions are LTI SSPs

It may be tempting to assume that the convolutional-SSPs are a restrictive subclass of convolutional models, since the convolutional-SSP was derived from unrolling the recurrent-SSP formulation. This is however not the case.

The section on LTI systems specifically noted that all continuous LTI SSM's have a continuous convolutional formulation (see equation (17)). Conversely, **[Williams and Lawrence 2007]** showcase that any continuous convolution filter $K(t)$ that is a rational function of degree N can be represented by an SSM of state dimension N. Thus, as the state size $N \to \infty$, the continuous convolutional-SSP can approximate any continuous convolution filter (section 2.4.3 in [31]), implying that the (continuous) convolutional-SSP can express any (continuous) convolutional model.

6.7 The Implications of the SSP Being Linear

The traditional RNNs are non-linear in their recurrence (see equation (8)). The fact that SSPs are linear in their recurrences may raise concerns in regard to the expressivity of the overall deep state space model. This section briefly states three arguments, each providing evidence as to why the linearity of the SSP does not imply reduced expressivity of the overall deep model.

- It was previously shown how the gating mechanism of a simple RNN-recurrence can be derived through discretization. This connects the SSP to a gated RNN-recurrence and suggests a notion of gating (and nonlinearity) may be implied by the discretization process.
- Drawing parallels between deep state space models and CNNs further supports the notion that linearity of the SSP does not constrain the overall expressivity of the deep state space model. Consider that CNNs have never been considered inexpressive and remain competitive in for instance computer vision with models such as ConvNext[53]. A common denominator for CNNs is the fact that their defining primitive are entirely linear kernel operations. Meanwhile, the overall CNN is made nonlinear through the incorporation of nonlinearities in the depth direction of the network architecture. Evidently, deep state space models should similarly be able to achieve their nonlinear modeling capabilities through the depth dimension of the network architecture.
- In the work **[Orvieto et al. 2023]** they prove that linear diagonal recurrences interleaved with multilayer perceptrons lead to arbitrarily precise approximations of causal sequence-to-sequence maps. Their proof relies on a separation of responsibilities. That is, the linear recurrence provides a lossless encoding of the input sequence, while the multilayer perceptron performs non-linear processing on the encoding. Their results demonstrate theoretical unrestricted expressivity of deep linear recurrent models. A deep state space model is exactly a deep linear recurrent models.

6.8 The State Space Primitive Summarized

This section attempts to give a combined and compact summary of the SSP. In this sense, this section may be considered a *conclusion* to the overall theoretical derivation and interpretation of the SSP.

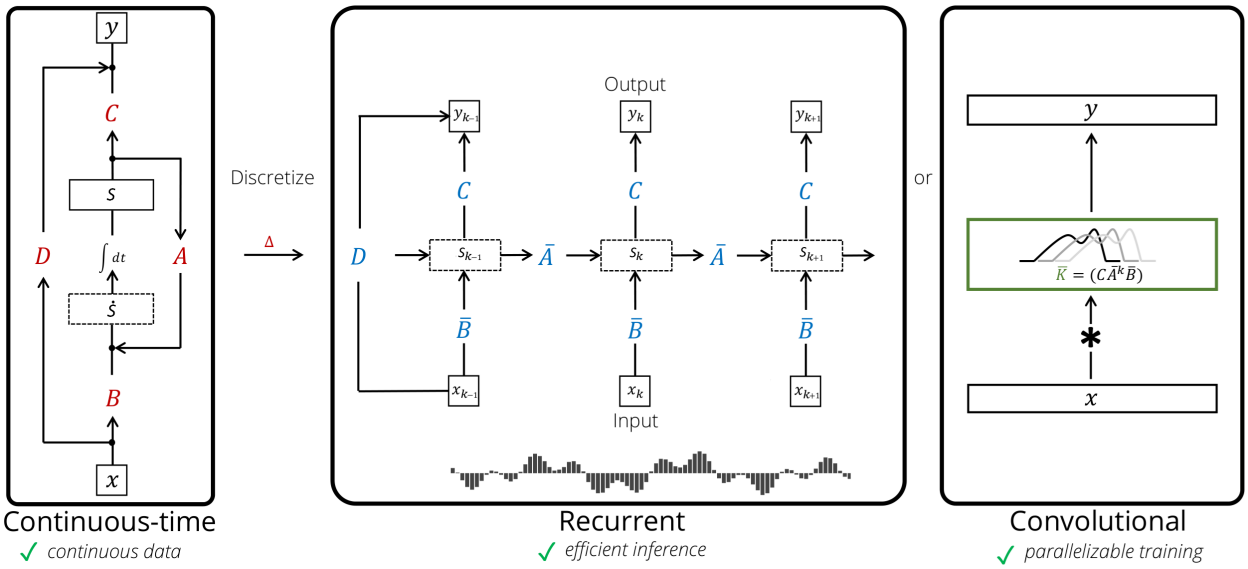

Figure 5: The SSP, in continuous, recurrent and convolutional mode. (The figure is a modified version of figure 1 from [35]).

The overarching notion of the SSP is to simulate the discrete sequence as a continuous time SSM. The SSM is thought to represent an N'th order (rewritten as N first order) differential equation that models an underlying latent continuous time function that fully describes the observed sequence. Evidently, the goal is to simulate the observed data according to the true underlying function using the associated continuous time parameters $A(t), B(t), C(t)$.

It is immediately assumed that the parameters are constant across time (A, B, C), that is, time invariant. As a result the SSM emerges as a particular well behaved LTI function-to-function map, that may be expressed as a continuous convolution. To bridge the gap between the latent parameters (A, B) that describe the continuous-time system and the observed discrete sequence, the latent parameters has to be discretized according to a step size Δ. Applying zero order hold discretization to the continuous SSM produces an exact recurrent formulation with discretized parameters, $(\bar{A}, \bar{B})$. Subsequently the recurrence may be unrolled to produce a corresponding discrete convolution formulation.

The discrete convolution formulation allows for the sequence to be simulated by a single global convolution operation. By utilizing the convolution theorem, the global convolution can be evaluated in $O(L\ log(L))$, thereby allowing for efficient training. Afterwards, at inference time, if faced with the task of auto-regressive generation, the SSP may be employed in its recurrent formulation for efficient inference. As a result of the dual recurrent-convolutional formulation, the SSP exhibits both efficient training and efficient inference.

Since the parameters (A, B) are stored in their continuous time representation, it allows for Δ to be a modulating parameter that allows for the time scale of the data to be learned. The discretization of A and B merely becomes the first step of the computation graph of the SSP. Furthermore, the role of discretization can be framed as a learned gating mechanism, corresponding to that of a simple gated RNN-recurrence.

Finally, it was argued that while SSPs are fully linear, it does not compromise expressivity when incorporated into deep sequence model: The deep sequence model may simply achieve its nonlinearity through the depth

direction of the network architecture similar to CNNs.

7 Computationally Feasible State Space Primitives

The SSP formulations derived in the previous section are not yet suitable to be used in a deep sequence model as certain computational aspects renders them infeasible. It was established that the convolutional formulation of the SSP enables log linear scaling in the sequence length. While the convolution computation itself is efficient, constructing the kernel used for the convolution still constitute a computational bottleneck. This section will introduce the 3 practical SSP variants (s4[34], s4d[36] and s6[32]) each of which addresses the computational bottleneck making them readily applicable in a deep sequence model.

7.1 Dimensionality of the SSP

The solution to the computational bottleneck partly lies in careful choice of dimensionality in regard to the SSP. When the generic sequence primitive was introduced in section 2 it was defined as a $\mathbb{R}^{L\times d} \to \mathbb{R}^{L\times d}$ transformation. Nonetheless, the SSP was deliberately introduced as an $\mathbb{R}^{L\times G} \to \mathbb{R}^{L\times G}$ transformation. The discrepancy is a consequence of enhanced flexibility in the SSP setting. The formulation allows G to be any number which has exact division with d to produce a whole number of heads $\mathcal{H}$.

$$\mathcal{H} = \frac{d}{G} \tag{52}$$

Consequently the token embedding dimension d is split into $\mathcal{H}$ sub-feature vectors of size G. Each sub-token-embedding-vector is processed by an independent SSPs to produce $\mathcal{H}$ different outputs. The $\mathcal{H}$ outputs are finally concatenated to reproduce the full token embedding of size d. This recipe is entirely analogous to the concept of multihead-self-attention[80] and commonly referred to as MIMO which stands for multiple input multiple output. One edge case of MIMO is denoted maximal-MIMO, which is when $d = G$ thus $\mathcal{H} = 1$ and analogous to singlehead-self-attention, that is the regular self-attention as defined in equation (7). The opposite edge case is called single input, single output (SISO), where $G = 1$ and there are as many heads as there are entries in d. The SISO formulation is of particular interest: Concretely each of the entries in the token embedding vector is processed independently by d independent SSPs. The d independent SSPs each have their own set of parameters (A, B, C, Δ). As a result the SISO SSP exclusively[2] performs temporal mixing (mixing along the sequence dimension), since the individual token embedding entries are treated by an independent SSP and thus unable to affect each other. From a theoretical stand point SISO implies simulating each feature of the token embedding as its own linear continuous time state space. Finally note that, as a consequence of the SISO formulation feature mixing (mixing along the feature dimension) must occur elsewhere in the network architecture.

Traditionally, RNNs and CNNs are formulated as maximal MIMO. Some recent successful attempts[64][53] at modernizing the RNN and CNN architectures have adapted the SISO formulation.

7.1.1 Applying SISO to the SSPs

All 3 SSPs variants (s4, s4d, s6) introduced will make use of the SISO formulation. Recall that the global SSP convolution kernel is given by:

$$\bar{K} = \left(C\bar{B}, C\overline{AB}, C\bar{A}^2\bar{B}, \ldots, C\bar{A}^{L-1}\bar{B}\right) \tag{53}$$

Applying SISO formulation reduces $\bar{B}$ and C to $(N \times 1)$ and $(1 \times N)$ matrices, reducing their associated operations from matrix-matrix multiplication to matrix-vector multiplications. $\bar{A}$ however is unaffected by the SISO formulation and remains an $(N \times N)$ matrix. Consequently materializing the kernel still requires L matrix-matrix-multiplications, thus computational complexity $O(LN^3)$ since the exponent in $\bar{A}$ depends on L. While the SISO formulation reduces the computation by a constant factor, materialising the kernel remains prohibitively expensive. This observation motivates imposing structure on A ($\bar{A}$) to further reduce the cost of materializing the kernel.

[2] With some reservation in regard to s6, which will addressed in full upon its proper introduction in section 7.5.

7.2 The Diagonal A

The simplest structure which would immediately reduce the computational complexity is to have A and hereby $\bar{A}$ be diagonal matrices. A and $\bar{A}$ on diagonal form will not receive explicit notation. Instead notation overloading will be employed and it should be apparent from the context which variant of A is used. Notice that for all practical purposes s4d and s6 always employs the diagonal A whereas s4 employs a diagonal plus low rank (DPLR). Evidently, the structure of A can also be inferred based on which SSP is currently mentioned in the text. (I originally tried to use a different notation for the diagonal A but it only made the matter more confusing.)

$\bar{A}$ as a diagonal matrix allows the matrix power to be conducted element-wise and reduces matrix-vector products with both C and $\bar{B}$ to element wise multiplication. Evidently, the complexity of the overall kernel computation reduces to $O(LN)$. Letting the transition matrix A be diagonal considerably reduces the amount of parameters in the SSP. Concretely the amount of trainable parameters is reduced by a factor of N but this can be accommodated by an adequate increase in the token embedding dimension d or state dimension N.

Given the diagonal $\bar{A}$ and the SISO formulation the computation of the k'th kernel entry of $\bar{K}$ reduces to.

$$\bar{K_k} = \sum_{n=0}^{N-1} C_n \bar{A}_n^l \overline{B}_n \tag{54}$$

(Notice that $\bar{A}, \bar{B}, C$, are for all practical purposes vectors in the above formulation and thus indexed as such.) Computing all elements of the kernel can be restated as one cleverly formulated vector-matrix product by leveraging the Vandermonde matrix. The Vandermonde matrix of diagonal $\bar{A}$ for a sequence length L is given as.

$$\mathcal{V}_L(\bar{A}) = [\bar{A}^0, \bar{A}^1, ..., \bar{A}^{L-1}] = \begin{bmatrix} 1 & \overline{A}_0 & \overline{A}_0^2 & \dots & \overline{A}_0^{L-1} \\ 1 & \overline{A}_1 & \overline{A}_1^2 & \dots & \overline{A}_1^{L-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \overline{A}_{N-1} & \overline{A}_{N-1}^2 & \dots & \overline{A}_{N-1}^{L-1} \end{bmatrix} \tag{55}$$

(Again $\bar{A}$ is diagonal and thus indexed as a vector). The compact vector matrix computation of $\bar{K}$ emerges as:

$$\overline{K} = \left(\overline{B}^\top \otimes C\right) \mathcal{V}_L(\overline{A}) = \begin{bmatrix} \overline{B}_0 C_0 & \dots & \overline{B}_{N-1} C_{N-1} \end{bmatrix} \begin{bmatrix} 1 & \overline{A}_0 & \overline{A}_0^2 & \dots & \overline{A}_0^{L-1} \\ 1 & \overline{A}_1 & \overline{A}_1^2 & \dots & \overline{A}_1^{L-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \overline{A}_{N-1} & \overline{A}_{N-1}^2 & \dots & \overline{A}_{N-1}^{L-1} \end{bmatrix} \tag{56}$$

Where $\otimes$ is element-wise multiplication. The naive Vandermonde matrix-vector multiplication has complexity $O(LN)$, but due to their structure, efficient $O(log^2(N+L)(N+L))$ algorithms[9] are available.

7.2.1 Expressivity of the the Diagonal A

Imposing structure on A raises a natural question regarding the consequences with respect to expressivity of the simplified SSP. This question can be dismissed through a short mathematical consideration regarding linear algebra. The source material for the derivation is the work of **[Oppenheim and Verghese, 2010]**.

Consider the continuous LTI system back from equation (13). For convenience it has been repeated below:

$$\begin{aligned} \dot{s}(t) &= As(t) + Bx(t) \\ y(t) &= Cs(t) \end{aligned} \tag{57}$$

Introduce the change of variable $s(t) = Mz(t)$. Choose M such that it is constant and invertible. The change of variable then corresponds to a change of coordinates to a new state vector $z(t)$. Rewriting in terms of the change of variables yields:

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{d}t}(Mz(t)) &= AMz(t) + Bx(t) \\ y(t) &= CMz(t) \end{aligned} \tag{58}$$

M being constant can be moved outside the differentiation on the left hand side. Subsequently M being invertible allows for M^{-1} to be multiplying through in the first equation.

$$\begin{aligned} \dot{z}(t) &= M^{-1}AMz(t) + M^{-1}Bx(t) \\ y(t) &= CMz(t) \end{aligned} \tag{59}$$

The model defined above is equivalent to (57) as M allows for an invertible transformation between the s and the z state. From the perspective of the input $x(t)$ and the output $y(t)$ the models are identical. Furthermore, $M^{-1}AM$ and A have identical eigenvalues implying that the natural frequencies of the systems are identical.

The significance of the result lies in the fact that $M^{-1}AM = \Lambda$ corresponds to a decomposition of A into a diagonal matrix Λ and displays that the Λ causes the same transformation as the original A albeit in different coordinates. The models are said to be similar and the change of coordinates is denoted a similarity transform.

This result justifies letting A be diagonal from the beginning since expressiveness is not compromised as demonstrated by the similarity transform.

Finally it is highlighted that while the diagonalization of the real valued A is always possible (up to an arbitrary small perturbation) it amounts to a complex valued decomposition unless A is symmetric.

7.3 s4d: (Diagonal) Structured State Space Sequence Mode

The previous section on the SISO formulation and the section on letting A be a diagonal matrix are the 2 ingredients needed to produce the first practical SSP variant. This SSP variant is dubbed s4d[36] in the literature. The 4 s's stands for structured state space sequence model and the d indicates A is a diagonal. Due to its SISO formulation s4d processes the token embedding sequence X by the means of simulating d independent continuous state spaces. That is employing d independent SSPs, one for each feature in the token embeddings. The d independent SSPs have their own set of parameters (A, B, C, Δ). The d different SSPs implies d different convolution kernels $\bar{K}$. These are constructed efficiently by applying equation (56), and relies on the Vandermonde product.

7.4 s4: Structured State Space Sequence Mode

The next concrete SSP variant of interest is termed s4. As the name suggests s4 is a more involved version of the s4d in which the A matrix is not on diagonal form. Consequently the convolution kernel needs to be constructed in a manner different from equation 56, as equation 56 relied on A being diagonal. Although the modification is very subtle on paper it entails a vastly more complicated approach for efficient construction of the kernel.

Historically speaking s4 is the predecessor to s4d. Although s4d is much simpler it was disregarded initially during the development of s4 as the diagonal formulation was not supported by HiPPO theory. HiPPO is the theoretical foundation on which SSP were originally developed. It is a mathematical framework

for optimal online function approximation. HiPPO is beyond the scope of this thesis but will be covered superficially in section 9. The specifics of HiPPO is secondary to understanding the relation between s4 and s4d. All the reader needs to know is that HiPPO theory prescribes the A and B matrix to have a particular parameterization. A as a diagonal matrix violates HiPPO theory and may at most be considered an approximation, and explains why diagonal A variants were initially disregarded. s4 features a formulation of A which is compliant with HiPPO.

The A matrix utilized by s4 is still required to be highly structured to maintain computational feasibility with respect to the construction of the kernel. Concretely s4 manages computational feasibility by representing A by diagonal plus low-rank decomposition (DPLR). HiPPO-theory proposes that the DPLR representation utilised in s4 should amount to a better memory properties of the state space than the diagonal representation used by s4d. The authors of the s4d paper displays that this theoretical advantage translates to s4 being slightly better than s4d empirically[36] in most instances. The DPLR looks as follows:

$$A \approx \Lambda - QQ^\top, \quad \Lambda \in \mathbb{C}^{N\times N}, \quad Q \in \mathbb{R}^{N\times 1} \tag{60}$$

Where Λ is a diagonal, Q is column vector such $QQ^\top$ form an outer-product, i.e. the low rank term. (This representation of A and the DPLR decomposition is slightly elaborated upon in section 9 but is mostly beyond the scope of this thesis similar to the rest of HiPPO-theory.) It is implicitly understood that when A is mentioned in the context of s4, the reader should think of the DPLR representation of A given above.

From this point, the materialization of $\bar{K}$, requires multiple highly non trivial tricks and techniques. The exposition of the involved steps will be kept superficial for 2 reasons: The authors themselves recognize the severely complicated nature of their method and highlight the fact that in practice s4 is only marginally better than s4d. Furthermore, the kernel construction used in s4 is not related to s4d nor s6 (the final SSP covered in next section) and insights do not carry over. Consequently, this thesis will only cover the intuition and implications of each of the necessary steps. Using the (HiPPO-theory compliant) DPLR representation of A the kernel may be materialized efficiently by employing the 4 steps:

- The first step recasts the kernel in terms of a generation function with respect to its coefficients. Intuitively this converts the kernel filter from time domain to frequency domain. Ultimately, the kernel is recovered by evaluating the generating function in particular points z. These points are chosen to be roots of unity[8] such that the matrix power $\bar{A}^k$ evaluates to 1 and essentially vanishes. This step gets rid of the costly matrix powers in $\bar{A}$, but entails a formulation involving matrix inversion of $\bar{A}$.
- The next step is to mitigate the matrix inversion created by the previous step. In order to do so the woodbury matrix identity[10] (which relates to inversion of DPLR matrices) is employed. In essence this step converts the inversion of $\bar{A}$ in the generation function to only be in terms of the diagonal $\bar{A}$ and not the low-rank term $QQ^\top$, making the inversion trivial.
- The rewriting in the previous step leads to a formulation which involves expressions that are identical to cauchy matrices[3]. Cauchy matrices are a well studied subject with highly efficient matrix-vector multiplication algorithms with $O(log^2(N+L)(N+L))$ complexities rather than naive $O(NL)$. The implications being that some of the operations from the previous steps may be recast in terms of black box cauchy algorithms to reduce the overall computational complexity.
- The motivation to recast the problem in terms of a generation function was that eventually (now) the kernel could be resampled at very specific points, i.e. roots of unity such certain terms would vanish. Thus the final step is to sample the function and apply the inverse FFT in order to reconstruct the kernel in time domain.

The above procedure does not utilize approximations and thus produces an exact kernel. Moreover the procedure has computational complexity $O(log^2(N+L)(N+L))$, which is identical to the s4d case. Their dominating operations are respectively Vandermonde and Cauchy matrix-vector products which are closely

related and consequently the two methods have identical complexities. It should be noted that the s4 has a larger constant term.

7.4.1 Summarizing s4

The difference between s4d and s4 amounts to how A is parameterized and subsequently how the kernel $\bar{K}$ are efficiently materialised. Equivalent to s4d, s4 uses the SISO formulation by which it employs d independent SSPs. The A's are not diagonal but instead represented by the DPLR formulation from equation (60). Lastly the d kernels $\bar{K}$ are constructed using the above described 4 step procedure.

7.5 s6: Selective Structured State Space Sequence Model Computed with a Scan

The final SSP variant is termed s6. The s6 abbreviation is derived from "selective structured state space sequence model computed with a scan". s6 is a sizable departure from the previous models s4/s4d and the general SSP theory introduced thus far. The additional 2s's implies the extension over previous iterations and are related to selectivity and scans. Selectivity is related to how s6 breaks time invariance property to enhance expressivity. As a result the s6 SSP is no longer time invariant and the convolutional representation becomes infeasible. Scan refers to the new approach employed to maintain computational feasibility given the absence of the convolution formulation. This section will cover the motivation for the departure from time invariance and subsequently how time variance is incorporated into a state space primitive. The section will not address the details of how the s6 SSP archives computational feasibility through scans. Computational feasibility and scans is covered in depth in section 8.

7.5.1 Selectivity for optimal context compression

The authors of the s6 paper motivates the selection mechanism from a context compression perspective. The essence of causal sequence modeling is the ability to relate the current token to the context of all previous tokens. Different modeling approaches come down to a trade off between effectiveness and efficiency. In this regard the transformer is completely effective and totally inefficient, since it does not compress context but maintains the full history of the sequence. Keeping track of the full history becomes prohibitive for modeling certain modalities and for autoregressive generation and is the source of the quadratic scaling. On the contrary the state-full model will always be efficient as the recurrent state update has constant computational complexity. However the state-full models have confined effectiveness as they are limited by how well the state encapsulates the whole context. (Increasing the hidden state dimension s, increases the models ability to capture and store relevant context, with some penalty to efficiency, but ultimately the model remains efficient.) In other words the state-full models are limited by their ability to compress and represent the context of a given sequence, as such they must be particularly careful about what information they choose to store in the state. This fact motivates the adoption of a contextual selection mechanism to enforce the notion that only relevant information is allowed into the state and out of the state.

Aside from this intuitive and theoretical observation, the necessity for a selection mechanism is also supported through experimental observations. The authors of s6 empirically display that certain synthetic machine learning problems which require content aware reasoning, namely "selective copying" and "induction heads" are particularly difficult for the s4/s4d models. After equipping the SSPs with a selection mechanism (turning them into s6) these synthetic problems are solved perfectly. Moreover the s6 maintains perfect performance at test time even when faced with sequences orders of magnitude longer than what was present at training time.

7.5.2 Incorporating Selectivity

s6 employs the SISO formulation. Similar to s4d, s6 utilizes the diagonal state transition matrix A. The remaining state space parameters (B, C, Δ) receive a new parameterization reflecting the selection mechanism. Concretely the selection mechanism emerges as a result of letting B, C and Δ be input dependant. Their

input dependant variants will be denoted B_k, C_k and Δ_k. The input dependant state space recurrence then materialise as:

$$\begin{aligned} s_k &= \bar{A}_k s_{k-1} + \bar{B}_k x_k \\ y_k &= C_k s_k \end{aligned} \tag{61}$$

(Notice A_k is only input dependant as a result of the discretization procedure with the input dependant Δ_k)

Input dependant variants B_k, C_k, and Δ_k are archived by letting each of the parameters be functions of the current token embedding x_k. In the instance of B_k and C_k the functions which produce them are simply 2 different linear projections. The associated weight matrices are denoted $W^{(B)}$ and $W^{(C)}$ respectively and both have dimension $d \times N$.

$$\begin{aligned} B_k &= W^{(B)}\left(x_k\right) \\ C_k &= W^{(C)}\left(x_k\right) \end{aligned} \tag{62}$$

(As per usual the superscript in W simply denotes a distinction and not a mathematical operation). Notice that although s6 uses the SISO formulation only one single B_k and one single C_k are produced and not d. Evidently the same B_k and C_k are reused across all d different state spaces and some dependent relation between the d different state spaces occurs. This implies that while s6 is SISO, the different features of the token embeddings do not get treated independently as it is the case with s4 and s4d. In other words s6 entails some amount of feature mixing.

The input dependant function which produces Δ_k is slightly more complex. For clarity it is split into 3 intermediate steps. Each intermediate step is denoted δ and the shape of δ after each step is shown. The shape of the linear projections matrices $W^{(1)}$ and $W^{(2)}$ can be inferred based on the shape of δ. The steps are:

$$\begin{aligned} &\delta \leftarrow W^{(1)} x_k, && \delta \in R^1 \\ &\delta \leftarrow \texttt{REP}(\delta), && \delta \in R^d \\ &\delta \leftarrow W^{(2)}(\delta + b), && \delta \in R^d \\ &\Delta_k = \omega(\delta) && \Delta \in R^d \end{aligned} \tag{63}$$

The 4 operations explained in order. 1: The input x_k is projected down to a single scalar. 2: Said scalar is broadcast/repeated back to dimension d . 3: A d dimensional learnable parameter b is added element-wise and subsequently their result undergoes another projection. 4: Finally the soft-plus ω which is $\omega(\cdot) = \log(1 + \exp(\cdot))$ is applied which results in Δ_k. Notice that soft-plus produces a strictly non-negative output, thus enforcing that time scales are never negative.

Although the parameterization of Δ_k seems unnecessary complicated there are intuitions as why it looks as it does. The reason why the 1. step projects the input into a single scalar is based on the following observation: Consider an instance where the current token is pure noise and should not be allowed into the state. In this instance all d state spaces should ignore it. This notion is enforced by projecting the input x_k into a single scalar and subsequently broadcasting it back (step 2) to the adequate dimension of d.

Moreover the choice of using the soft-plus parameterization for Δ_k originates from the theoretical results discussed next.

7.5.3 The Origin of the Δ_k Parametrization

Recall the derivation in section 6.6.2 which displays how the simple gated recurrence emerges from discretizing a 1D linear continuous time state space model. Supported by this result it was argued that the discretization corresponds to a learnable gating mechanism. This result however, had a major shortcoming, constituted by the fact that the derivation assumed that the arbitrary expression z_k was input invariant. The selection

mechanism makes s6 inherently input variant, in turn removing the necessity for the assumption, thereby generalizing the result. The generalized result implied by the selection mechanism will now be derived. The source material for the derivation is appendix C of the s6 paper[32]. Consider again the 1D continuous time state space model with $A = -1$, $B = 1$

$$\dot{s}(t) = -s(t) + x(t) \tag{64}$$

Let the input dependant discretization parameter Δ_k be given by:

$$\Delta_k = \omega\left(f\left(x_t\right)\right) \tag{65}$$

Where f corresponds to the 3 intermediate steps of equation (63) and ω remains the soft-plus.

Applying zero order hold on the system (64) using the input dependant discretization parameter from equation (65) yields the discretized $\bar{A}$:

$$\begin{aligned}
\bar{A}_k =& \exp(A\Delta_k) \\
=& \exp\left(-1 \cdot \log\left(1 + \exp\left(f\left(x_k\right)\right)\right)\right) \\
=& \exp\left(\log \frac{1}{1 + \exp\left(f\left(x_k\right)\right.}\right) \\
=& \frac{1}{1 + \exp\left(f\left(x_k\right)\right)} \\
=& \sigma\left(-f\left(x_k\right)\right) \\
=& 1 - \sigma\left(f\left(x_k\right)\right)
\end{aligned} \tag{66}$$

Similarly B is discretized according to zero order hold to produced $\bar{B}$

$$\begin{aligned}
\bar{B}_k =& (A\Delta_k)^{-1} \cdot (\exp(A\Delta_k) - 1) B\Delta_k \\
=& -\Delta_k^{-1} \cdot (\bar{A}_k - I)\Delta_k \\
=& 1 - \bar{A}_k \\
=& 1 - 1 + \sigma\left(f\left(x_k\right)\right) \\
=& \sigma\left(f\left(x_k\right)\right)
\end{aligned} \tag{67}$$

Rewriting the continuous time state space model in terms of this discretization yields the gated RNN-recurrence:

$$\begin{aligned}
g_k &= \sigma\left(f\left(x_k\right)\right) \\
s_k &= \left(1 - g_k\right) s_{t-1} + g_k x_k
\end{aligned} \tag{68}$$

This time no reservations have to be made in regard to input Independence of the gate, as the selection mechanism allows for input dependence.

7.5.4 Input Dependant A

It is noted that A does not undergo a input dependent parametrization. The reasoning behind this choice stems from the fact that A only interacts with the state space through its discretization. Recall that the discretization of A is $\bar{A} = exp(A \cdot \Delta_k)$, and thus involves interaction with Δ_k which is already data-dependent. Consequently A is indirectly subject to data-dependence (selectivity) through the discretization process, and further data-dependency was deemed unnecessary by the authors.

7.5.5 Highlighting disparity between the parameter dimensionality of s4d and s6.

This short section is included to help conceptualise and understand the dimensionality of the state space parameters of s6 in comparison to s4d. In order to do so this section introduces the batch size which will consistently be denoted b.

Recall the s4d setting which had all parameters A, B and C be N dimensional vectors (since A is diagonal) with independent repetitions over the d SISO SSPs which in practice amounts to a combined dimensionality $(d \times N)$ for each of A, B and C. That is, independent of the sequence length L and the batch size b the parameters A, B and C always have shape $(d \times N)$.

Consider now the s6 setting. For **every single** token embedding in the batch b an N-dimensional projection B_k and C_k is produced. That is, when B_k and C_k are produced for a whole batch of shape $(b \times L \times d)$ the resulting shape of B_k and C_k are $(b \times L \times 1 \times N)$. Notice how B_k and C_k are no longer independent across the d SISO SSPs, but are instead shared for all dimensions d. Moreover the construction of Δ_k from a single token yields a d dimensional output such that, when materializing Δ_k over the whole batch the shape becomes: $(b \times L \times d \times 1)$.

Recall that A is not directly input dependant thus before discretization A has shape $(d \times N)$, while B_k and C_k has shape $(b \times L \times 1 \times N)$ and Δ has shape $(b \times L \times d \times 1)$. Subsequently at the time of discretization appropriate broadcasting (repetition) is applied across the necessary dimensions such that $\bar{A}_k$ and $\bar{B}_k$ receive full shape of $(b \times L \times d \times N)$. Notice C_k is unaffected by discretization and its shape remains $(B, L, 1, N)$.

The choice of letting $\bar{A}_k$ and $\bar{B}_k$ only fully materialize after discretization is related to hardware efficiency considerations. A full discussion regarding hardware aware consideration of the practical implementation of s6 can be found in section 8.

7.5.6 Summarising s6

Similar to s4d, s6 utilizes the SISO formulation and a diagonal A. The main distinguishing factor of s6 is the *selection mechanism* which materializes through letting B, C and Δ be input dependant. The input dependant B_k and C_k emerge as simple linear projections of the token embedding x_k. The input dependant Δ_k has a slightly more involved parameterization which involves multiple steps. The more involved parameterization attempts to reflect that if a token embedding x_k is pure noise it should be ignored by **all** the SISO states spaces. The selection mechanism entails 2 quantities. 1) B_k and C_k are shared across the d SISO state spaces thus the d state spaces may no longer be considered independent of each other. Evidently s6 does not exclusively perform temporal mixing but also implies some amount of feature mixing. 2) The selection mechanism breaks the time invariance property thus the global convolution may no longer be utilized for efficient training. Instead the recurrence is computed directly during training, which may be done in an efficiently manner by employing a parallel scan. The parallel scan is discussed in section 8.

7.6 An Overview and Comparison of s4, s4d and s6

This short subsection summarized and compares the 3 practical SSPs derived in the previous sections.

	A	Complex A	SISO	Feature Mixing	Linear	Time Invariant	Effcient Computation
s4	DPLR	✓	✓		✓	✓	Global Conv
s4d	Diag	✓	✓		✓	✓	Global Conv
s6	Diag		✓	✓	✓		Parallel Scan

Table 1: Comparison of the central properties of the SSPs. Empty field simply means *no*. The subject of complex valued A is briefly covered in section 9.

Table 1 highlights the primary features of s4 s4d and s6, showcasing both where they differ and where they are alike.

The table makes it immediately apparent, as previously mentioned, that s4 and s4d are very similar. They differ in how they parameterize A and as a consequently how they construct their respective kernels used for the global convolution operation. s4d uses a very simply procedure based on a single vector-matrix product with a cleverly constructed Vandermonde matrix. s4 requires a a vastly more complicated procedure, which interestingly have the same computational complexity as the simple Vandermonde approach of s4d. HiPPO theory proposes that the DPLR representation utilised in s4 should amount to a better memory properties of the state space than the diagonal representation used by s4d.

All 3 SSPs utilize the SISO formulation, and thus employ d state space, one for each of the token embedding features. All 3 SSPs are linear in parameters and fundamentally processes the data according to the same discrete state space recurrence given by:

$$\begin{aligned} s_k &= \bar{A}s_{k-1} + \bar{B}x_k \\ y_k &= Cs_k \end{aligned} \tag{69}$$

With the major distinction being that in the setting of s6 the transition matrix $\bar{A}$, the input matrix $\bar{B}$ and the output matrix C depends the current x_k. This is achieved by letting B_k, C_k and Δ_k materialise through parameterized functions of the current x_k. One consequence here off is that $\bar{B}_k$ and C_k are shared across the d different state spaces. Therefore, despite the SISO formulation, the d state spaces may no longer be considered independent of each other. Evidently s6 entails some amount of feature mixing, where as s4 and s4d do not. Another consequence is that s6 may no longer be considered *time invariant* thus the global convolution is disallowed. The above state space recurrence is instead computed directly as a recurrence during training.

8 Parallel Scans: Hardware Efficient Computation of Linear Recurrences

This section covers how s6 manages computational feasibility. Although the described concepts are in the context of the s6 SSP they conceptualize to linear recurrent models in general.

Recall that the input dependant B_k, C_k and Δ_k in the recurrence of s6 makes the global convolution computation employed in s4 and s4d infeasible. While the s6 recurrence is input dependent it still remains linear in the recurrence, which permits hardware efficient alternatives for computing the recurrence. Subject to certain reservations these alternatives are faster in theory: The naive recurrence requires$O(b \cdot L \cdot d \cdot N)$ FLOPs. On the contrary the FFT convolution requires $(b \cdot L \cdot d \cdot log(L))$ FLOPs and has a larger constant factor. Evidently, the recurrence is truly linear in sequence length, while the convolution is only pseudo linear. However, since the convolution does not materialize the hidden state it is independent of the state dimension N. In conclusion, for sufficiently long sequences and reasonably sized hidden state dimension the recurrence requires fewer FLOPs.

The theoretical amount of FLOPs are irrelevant if the computation cannot be parallelized to take advantage of the multiprocessing power of GPUs. Traditionally, RNNs have been difficult to optimize[63] and slow to train. Their speed limit lies in the nonlinearity in the recursion. The nonlinearity has been thought to be crucial for expressivity, but prohibits parallelization along the sequence dimension. Recent work[62] has found making the recursion entirely linear not only improves expressivity, but enables proper parallelization along the sequence dimension. The algorithmic ideas of parallelization of a linear recurrence dates back to 1980[47], but has only recently (+2017) been demonstrated in practice in the context of deep recurrent models [55][75]. Inspired by these, the s6 paper computes the recurrence using a work-efficient associative parallel scan.

The field of parallel scans is well studied[51][23] and throughout the years a number of different parallel scan algorithms have been proposed. Nonetheless the s6 paper is particularly vague in regard to exactly which underlying algorithm their implementation is based on, aside from calling it "work-efficient associative parallel scan". Neglecting the details related to exactly how the recurrences are computed in parallel scan appears to be a general trend in the linear RNN deep learning literature.[3] While linear RNN papers rarely address the details, there seems to be a particular parallel scan paper which consistently gets referenced. The paper is from 1990 and proposes an algorithm which today commonly goes by the name the Blelloch[13] scan. Moreover, the Blelloch scan also appears to be a popular choice in other fields. For instance it can be employed to parallelize certain sorting algorithms[37].

While it is unclear in the setting of s6, exactly which approach is employed, the Blelloch scan is a candid guess. Parallelization along the sequences lies at the heart of the success of modern RNNs and enables recurrent formulations beyond the LTI recurrences used in the s4 and s4d. Consequently this thesis deems some coverage of the underlying parallelization mechanism essential. Given its prevalence the obvious algorithm to cover is the Blelloch scan.

8.1 Computing a Recurrence in Parallel Using the Blelloch Scan

A linear recurrence is an instance of the broader class of computations known as scans. Subject to certain associative conditions a scan can be computed in parallel over the sequence length. (Associativity and the details of what qualifies a computation as a scan is secondary for now.) The Blelloch scan is an instance of an algorithm, which computes a scan operation of a sequence in parallel over the sequence length. For simplicity the Blelloch scan will be introduced in the simple context of the cumulative sum. The cumulative sum is an instance of scan computation and is often formulated as a simple recurrence.

$$X[k] = X[k] + X[k-1] \qquad X[-1] = 0, \quad k = 0..L-1 \tag{70}$$

[3] Either `jax.lax.associative_scan` is used or a custom cuda kernel. In either instance the practicalities are omitted. It almost seems associative scans are treated as fundamental operations which do not need explaining.

It somewhat resembles the linear SSP recurrence in equation 61 as such the cumulative sum is used to demonstrate conceptually how the Blelloch scan works.

8.1.1 The Blelloch Scan Demonstrated Using the Cumulative Sum

The Blelloch scan has two phases, an "upsweep phase" and a "down sweep phase". The Blelloch scan pseudocode in the context of cumulative sums can be seen below.

Algorithm 1 `UP-SWEEP (cumulative sum)`

`Input :` X `(Array of length L)`
`Output:` X `(Intermediary result to be used in DOWN-SWEEP)`

for d **from** 0 **to** $lg_2(L) - 1$ **do**
  for i **from** 0 **to** $n - 1$ **by** 2^{d+1} [`in parallel`] **do**
    $X\left[i + 2^{d+1} - 1\right] \leftarrow X\left[i + 2^{d} - 1\right] + X\left[i + 2^{d+1} - 1\right]$

See the comment below the `DOWN-SWEEP` algorithm.

Algorithm 2 `DOWN-SWEEP (cumulative sum)`

`Input :` X `(Array of length L, produced by UP-SWEEP)`
`Output:` X `(Cumulative sum)`

$T \leftarrow X[L-1]$
$X[L-1] \leftarrow 0$
for d **from** $lg_2(L) - 1$ **downto** 0 **do**
  for i **from** 0 **to** $n - 1$ **by** 2^{d+1} [`in parallel`] **do**
    $t \leftarrow X\left[i + 2^{d} - 1\right]$
    $X\left[i + 2^{d} - 1\right] \leftarrow X\left[i + 2^{d+1} - 1\right]$
    $X\left[i + 2^{d+1} - 1\right] \leftarrow X\left[i + 2^{d+1} - 1\right] + t$

$X \leftarrow X[1:L]$ `%python-style slicing, to get rid of the first element`
$X \leftarrow$ `concatenate`(X, T)

The outer loop corresponds to the height of the binary tree. [`in parallel`] implies that each iteration of the inner loop is distributed to a different processor. Subsequently (assuming $L/2$ processors) the inner loop is executed in constant time since one iteration is $O(1)$ and all iterations are run in parallel on a different processor. Evidently the total parallel time complexity of the algorithm corresponds to the outer loop which is exactly the height of the binary tree thus $O(log_2(L))$.

8.1.2 The `UP-SWEEP` Explanation and Example

The up-sweep phase computes the sum (reduction) of the array using a binary tree approach and stores all intermediate calculations for later use. The binary tree emerges by considering the elements of the array as the leaves of the tree. Then pairwise addition is performed on the leaves to produce a set of nodes. (See step 0 to 1 in the example below). The nodes are used as the new pairs and the process repeats until the root is reached. (Step 1 to 2 and subsequently 2 to 3 in the example). Once the up-sweep has completed, the root of the tree emerges as the final element in the array and contains the sum of the array. The remaining elements in the array do not yet constitute a cumulative sum.

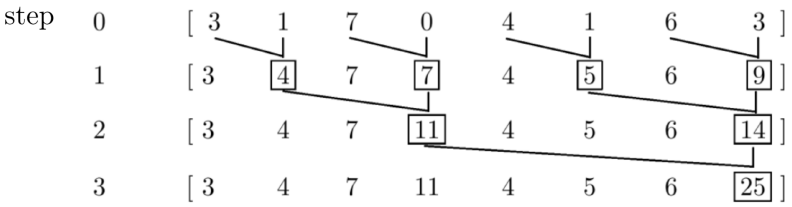

Figure 6: Example of the `UP-SWEEP` phase for computing cumulative sum. The black lines indicate addition between two entries and the black boxes indicate values produced by the addition. In unison the black lines form the binary tree. In practice the operations are performed in place.

8.1.3 The `DOWN-SWEEP` Explanation and Example

The down-sweep phase operates on the resulting array produced by the up-sweep phase, i.e. the array corresponding to step 3 of in the example. The first step of the down-sweep is to store the final element (the root of the up-sweep tree) and replace it with the identity which in the context of cumulative sums is 0. Subsequently, the binary tree is traversed from the root and down (Binary trees are upside down compared to actual trees). During the down-sweep phase the order of operations mirrors that of the up-sweep phase. However each addition operation is accompanied by a copy operation. Continuing the example from before yields:

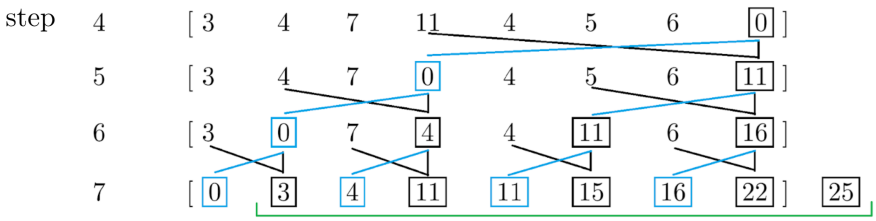

Figure 7: Example of the `DOWN-SWEEP` phase for computing cumulative sum. Black lines are additions and black boxes the result hereoff. Blue lines indicate a copy operation and blue squares are the result of said copy operation. In practice the operations are performed in place.

Once the down-sweep has completed the resulting array contains the cumulative sum of the original array, but is off by 1 element. Conveniently the missing element is exactly the root of the upsweep, which was stored.

The correctness of the algorithm can be proved using induction [13]. However, the intuition behind the copy operation in the down-sweep phase is opaque and the algorithm as a whole is confusing. Hopefully the example is enough to convince the reader that the algorithm actually computes the cumulative sum.

8.1.4 Efficiency of the Algorithm

The Blelloch scan algorithm is optimal for parallel scan problems in regard to parallel complexity. Note that parallel complexity is different from traditional big-O complexity. Parallel complexity is how the number of parallel steps scales with the input length under the assumption of infinitely many processors. Consider the simpler sub-problem of computing the sum of an array. The optimal parallel complexity is $O(log_2(L))$ and is achieved by employing a binary tree computation analogously to the up-sweep and down-sweep. Evidently employing a constant (2) number of binary trees to solve the problem of the cumulative sum must also be optimal in the parallel sense.

The Blelloch scan algorithm is work efficient. A parallel algorithm is work efficient if it does not perform asymptotically more work than the serial version. That is, disregarding constant factors they scale similarly in input size. Consider as an example serially computing the cumulative sum of an array. This serial

computation requires $L-1$ additions. Notice that the up-sweep in the example performs $L-1=8-1$ additions. Similarly the down-sweep performs $L-1=8-1$ additions. Disregarding reads and writes Blelloch scan performs a constant amount of $2\times$ additions of the serial version and thus it is work efficient.

8.1.5 Beyond Cumulative Sums

Although evident from the example that cumulative sums may be computed using parallel scans, it may not immediately be apparent how the parallel scan can be applied to the s6 state space recurrence. The Blelloch paper[47] demonstrates that any first order linear recurrence may be computed using a scan if the recurrent expression is associative. That is, if the recurrence computation can be expressed by some binary operator $\odot$ for which it holds that $(a \odot b) \odot c = a \odot (b \odot c)$. It is beyond the scope of this thesis to formulate the binary operator and proof associativity in the instance of the s6 recurrence. For a concrete example demonstrating this procedure, in the instance of a simple linear recurrence, the reader is advised to consult appendix H of [75].

8.2 Hardware Efficient Considerations of the s6 Implementation

While the exact details of the parallel scan implantation used in the s6 paper is nonexistent, the authors do however highlight some of their considerations in regard to how their implementation is optimized for hardware efficiency. Their approach is based on the general observation[19][26][43], that most operations on modern GPUs(apart from matrix-matrix-multiplication) are memory bound. That is, for most operations GPUs are limited by how fast they can move data between GPU-ram and GPU-cache, rather than how fast they can perform the actual computations.[4] Moving data between ram and cache is usually denoted input/output operations or i/o for short.

Inspired by the fact that the global convolution employed in s4 and s4d do not materialize the states, they employ a similar approach. Recall that materializing the full state of an entire sequence yields a tensor of dimension $(b \times L \times d \times N)$. (The term full state will be used to denote anything of shape $(b \times L \times d \times N)$). Moreover, recall that the discretized input dependant parameters $\bar{A}_k$ and $\bar{B}_k$ also are of size $(b \times L \times d \times N)$. Only a fraction of such a large amount of data fit in the cache of the GPU and constant data transfer would be required. Recall that the non discretized version of A only has dimension $(d \times N)$ and although input dependent the non discretized B_k, C_k and Δ_k have reduced dimension $(b \times L \times \mathit{1} \times N)$, $(b \times L \times \mathit{1} \times N)$ and $(b \times L \times d \times \mathit{1})$ respectively. These notions of dimensionality of the different quantities led to the following implementation considerations:

- **Kernel fusion**: The s6 kernel is implemented such that it reads the non discretized parameters from GPU-ram and performs the recurrence including the discretization step and output projection with C in one fused operation. In this way the state is only momentarily materialized when required and the GPU reads and writes at most data of size $(b \times L \times d)$. Their fused kernel implies a reduction in i/o operations proportional to the state size N. As a result they observe that their i/o aware fused kernel amounts to a speed up of 20-40 times in practice.
- **Blocking**: In instances where $(b \times L \times d)$ is larger than the GPU-cache the sequence is split into adequately sized blocks along the sequence dimension and processed sequentially. After each block has been processed the last state of the recurrence is returned and supplied as the starting point for the parallel scan of the next block.
- **Recomputation**: Usually the full computation graph and all intermediate values are stashed (saved in memory) during the forward pass of a deep model, as they are needed during the backward pass to compute the gradients. Recomputation is an approach where intermediate values are not stored but instead recomputed during the backward pass when they are needed. In the implementation of s6 the states are only present as long as they are required, nonetheless the states are needed to compute the gradients during backpropagation. To avoid having to store the states during the forward pass,

[4] GPU-ram is usually referred to as GPU-HBM and GPU-cache as SRAM, but for simplicity and similarity the broader terms ram and cache have been borrowed from CPU terminology.

which would induce a large i/o overhead, they simply recompute them during the backward pass. This approach not only saves memory, it is also faster.

9 Does State Space and HiPPO Theory Matter?

This section first briefly introduces HiPPO theory on a conceptual level. Then it will be argued (based on the findings of **[Orvieto et al. 2023]**) why HiPPO theory and SSP-theory may be of less importance than initially thought. Finally the use of complex number in modern recurrent models are briefly mentioned.

9.1 HiPPO: Higher Order Polynomial Projection Operators

This section will briefly introduce the theoretical concept of optimal online memorization (HiPPO-theory), which was originally thought to be one the key sources to the success of the SSP based models over conventional RNNs. Subsequently this section will elaborate why HiPPO-theory might be of less importance than initially thought. Although ambiguity remains in regard to the importance of HiPPO-theory, HiPPO-theory itself has been fundamental for the development SSP field as a whole. Moreover HiPPO-theory prescribes the initialization schemes of A and B in all SSPs employed in the experimental section of this work. Mathematical derivations are in this section omitted as the focus is a brief entirely conceptual overview.

In the HiPPO paper[33] the authors identify the core challenge of recurrent models as: *Maintaining a representation of the entire cumulative history of a sequence using bounded storage*. A representation which must be updated gradually as more data is revealed. HiPPO-theory frames this challenge as an optimization problem and demonstrates how to derive, subject to specific preference choices, optimal closed form solutions. The HiPPO acronym stands for Higher order Polynomial Projections Operators, and reveals some of the key ingredients. The overarching goal of HiPPO is to online maintain a compressed representation of the history of some function, by means of storing its optimal coefficients in some basis.

The essence of HiPPO is expressed by the following optimization problem. Given an input function $f(t) \in \mathbb{R}$ on $t \geq 0$ the goal is to memories the history of $f(t)$ up until the current time t. This will be done approximately, by determining the $g^{(t)} \in \mathcal{G}$ which minimizes:

$$\left\| f_{\leq t} - g^{(t)} \right\|_{L_2\left(\mu^{(t)}\right)} \tag{71}$$

Each component of this formulation will now be described conceptually.

- $f_{\leq t}$ is defined as $f_{\leq t} := f(x)|_{x \leq t}$. The definition is meant to indicate the *cumulative history* (the infinite number of function evaluations) of $f(x)$ up until the current time t.
- The function space $f_{\leq t}$ resides in is denoted $\mathcal{F}$ and is intractably large. Thus instead N dimension subspace $\mathcal{G}$ is considered . The basis chosen to represent the subspace $\mathcal{G}$ are the orthogonal polynomials (the Legendre polynomials to be specific). $\mathcal{G}$ defines the scope of the allowable approximations. Since $\mathcal{G}$ is N dimensional, the basis are the polynomials of degree at most N. In this sense N determines the magnitude of the compression or the precision of the approximation.
- $g(t)$ is the current approximation of $f_{\leq t}$. $g(t)$ is expressed in terms of its N coefficients $c(t)$ of the polynomial basis $\mathcal{G}$.
- $\|\cdot\|_{L_2\left(\mu^{(t)}\right)}$ is a particular norm used to quantify the magnitude of the error of the current approximation. Minimizing this norm corresponds to finding the optimal approximation.
- The characteristics of the norm depend on the term $\mu(t)$, which is a time dependent probability measure. Consequently $\mu(t)$ governs the importance of different aspects of the norm. That is, the choice of $\mu(t)$ determines which part of the cumulative history of f is emphasized in the approximation. The details regarding how $\mu(t)$ affects the norm is related to square integrals, inner products and Hilbert spaces and is beyond the scope and interest of this thesis.

In words: For any function f at every time t there exists an optimal projection $g(t)$ of f onto the N dimensional subspace $\mathcal{G}$ subject to the measure $\mu(t)$ weighting the past. Using the appropriately chosen orthogonal

polynomials as basis produces the N coefficients $c(t)$. The N coefficients $c(t)$ constitute a compressed history of f while satisfying linear dynamics.

HiPPO is a 2 fold operator which first performs the projection onto $\mathcal{G}$ and subsequently extracts the coefficients $c(t)$.

$$c(t) = \text{coef}_t\left(\text{proj}_t(f)\right) \tag{72}$$

It can be shown that the above coefficient function for some $A(t)$ of dimension $N \times N$ and B(t) of dimension $N \times 1$ satisfies the ODE:

$$\frac{\mathrm{d}}{\mathrm{d}t}c(t) = A(t)c(t) + B(t)f(t) \tag{73}$$

Implying that $c(t)$ can be obtained in an online manner by solving an ODE. Notice this is entirely analogous to the continuous state space models described in previous sections. The specific choice of using orthogonal polynomials as the basis is what entails the linear dynamics and subsequently amounts to a linear system of ODEs of the form (73).

Framing the problem of memory as optimal online function approximation allows for the construction of memory mechanisms with specific theoretical properties by simply choosing the measure function $\mu(t)$ appropriately. Based on this notion the HiPPO paper proposed a variety of measure functions resulting in different memory mechanisms. Their most performant approach is dubbed scaled Legendre measure (LegS). In contrast to traditional signal processing approaches which often employ a fixed sliding window, the LegS approach follows the intuition that in order to avoid forgetting the window size should be adaptive. Consequently, the measure should assign uniform weight to the entire history of the function f:

$$\text{LegS}_\mu = \mu(t) = \frac{1}{t}\text{I}_{[0,t]} \tag{74}$$

Employing the LegS measure produces the continuous and discrete systems:

$$\begin{aligned} \frac{d}{dt}c(t) &= -\frac{1}{t}Ac(t) + \frac{1}{t}Bf(t) \\ c_{k+1} &= \left(1 - \frac{A}{k}\right)c_k + \frac{1}{k}Bf_k \end{aligned} \tag{75}$$

From which closed form formuls for a A and B and be derived.

$$A_{LegS} = \begin{cases} (2n+1)^{1/2}(2k+1)^{1/2} & \text{if } n > k \\ n+1 & \text{if } n = k, \\ 0 & \text{if } n < k \end{cases} \qquad B_n = (2n+1)^{\frac{1}{2}} \tag{76}$$

The above matrix $A_{LegS} \in \mathbb{R}^{N \times N}$ is called the HiPPO-LegS matrix.

The authors showed that simply letting the sequence primitive of some deep model be given by the discrete recurrence in (75), which uses non-trainable A and B instantiated according to (76), allowed them to outperform conventional RNNs.

In subsequent papers novel approaches in regard to computational feasibility (s4 and s4d) allowed for A and B to be trained alongside the rest of the parameters in the network. However, it was displayed[36] that if A and B were not instantiated according to (76) or similar HiPPO derived approaches the models were substantially less performant. As a result all subsequent SSP-based models are initiated according to HiPPO theory, typically the LegS approach.

9.2 HiPPO-LegS decomposition and Initialization Scheme

All SSPs based models employed in the experimental section of this thesis will use the HiPPO-LegS prescribed initializion schemes of the transition matrix A and input matrix B from equation (76). However, the HiPPO-LegS matrix is not used directly but is decomposed into a the sum of a normal and rank-1 matrices, which may be unitarily conjugated into a (complex) diagonal plus rank-1 matrix. That is, the HiPPO-LegS matrix can be *represented* as a complex diagonal plus low rank (DPLR) formulation as follows:

$$A_{LegS} \approx \Lambda - QQ^{\top}, \quad \Lambda \in \mathbb{C}^{N\times N}, \quad Q \in \mathbb{R}^{N\times 1} \tag{77}$$

(Notice the HiPPO-LegS matrix is not "equal" to the above DPLR expression. The details are convoluted and uninteresting thus omitted. See [36] for the specifics regarding the representation.) Where Λ is a complex diagonal, Q is column-vector such QQ^{T} form an outer-product, i.e. the low rank term.

Notably the full DPLR representation of the HiPPO-LegS is the one employed by s4 and used to initialize its transition matrix A. In the setting of s4d only Λ is used to initialize A. Finally, in the setting of s6, $\mathbb{R}(\Lambda)$ is used to initialize A, that is the real part of Λ.

9.3 Questioning SSP and HiPPO Theory

This section is based on the findings of a recent 2023 paper. This paper which among other things, questions the importance of HiPPO theory for SSPs is called "Resurrecting Recurrent Neural Networks for Long Sequences"[62]. The paper is from Google Deepmind but is coauthored by Albert Gu the founding father of both HiPPO-theory and the SSP-line of work. Moreover, the paper has turned out to be a precursor for Google's very recent state of the art recurrent model[20] which matches and exceeds the performance of s6.

The authors ask the question: "Can we match the performance and efficiency of deep continuous-time state space models using deep RNNs?" Here they concretely refer to deep state space models based on the s4 and s4d SSP.

Their approach to answering the question departs from the complicated ODE, state space and discretization theory and instead applies a simpler first principled methodology with respect to linear algebra and deep learning. Their point of origin is a simple maximal-MIMO conventional RNN. From there they identify and subsequently modify 4 key areas of their conventional RNN, after which they achieve performance and computational efficiency similar to SOTA SSPs based models. The 4 key areas are:

- They too find that the recurrence of the (hidden) state should be entirely linear as it both improves model expressiveness (accuracy) and computational parallelism.
- Similarly to s4 versus s4d they find that the dense linear layer in the recurrence can be parameterized as a diagonal matrix without sacrificing expressiveness of the model. Moreover, this modification allows their RNN to match the computational speed of the state space models. Contrary to the global convolution used in s4/s4d but similar to s5 and s6 their RNN utilized a parallel scan to compute the recurrence.
- They acknowledge the importance of the initialization of the diagonal recurrence matrix (similar to A in an SSP) both in terms of stability and learning long range dependencies. However, rather than initializing the diagonal according to specific HiPPO prescribed structure, they propose a novel initialization scheme. Their proposal is derived as a diagonalization of the widely used Xavier initialization[28]. Their initialization scheme is entirely unstructured and involves uniform sampling of the complex unit disk. Moreover, they propose a simple exponential parametrization of the diagonal recurrence matrix to enforce stability during training.
- Finally they employ a novel normalization scheme to the inputs of the recurrence, that in SSP terminology they scale the term Bx_k before it is added to the state.

While the 2 first modifications are mostly related to computation efficiency and are similar to techniques employed by state space models, the 3. and 4. modifications are a distinct deviations from SSPs. Due to the

success of their alternative and novel initialization of the recurrent parameters, they propose the deposition: "It is the eigenvalue distribution of the recurrent layer at initialization that determines ability to capture long range dependencies". Hereby they insinuate that there is nothing special about the HiPPO initializing apart from the fact that it coincidentally imposes an adequate eigenvalue distribution on the state transition matrix A.

Additionally, they investigate the role of discretization (zero order hold) and the Δ parameter used in SSPs. Here they derive that discretization performs normalization similarly to their proposed normalization scheme. The implications being that, there is nothing magical about discretization and the reason it is important is due to its simple normalization properties.

A final implicit take away from their work which they do not highlight themselves is: Their model uses a maximal-MIMO formulation, which may be considered the opposite of the SISO formulation employed in all 3 introduced SSPs. Evidently, from a modeling perspective maximal-MIMO may be just as effective as the SISO formulation.

9.4 The Use of Complex Numbers in the Transition Matrix of Deep Recurrent Models

The modern deep recurrent literature is ambiguous in regard to using complex numbers in the state transition matrices. Among both SSP based and non SSP based deep recurrent models there seems to be a recent trend towards non complex valued parametrizations. Thus the subject of complex numbers adds another layer of complexity while not applying to all settings and therefore will be kept to a minimum in this thesis. However, the following is noted in regard to the use of complex numbers in the state transition matrices:

The state transition matrix of the recurrence employed in the Linear-Recurrent-Unit[62], s4 and s4d are complex valued. The use of complex numbers is motivated from the observation that: In all 3 instances the employed state transition matrix is either thought to represent a diagonalization or a DPLR-decomposition of some real valued and dense transition matrix as demonstrated in section 7.2. For this diagonalization/decomposition to hold for any A (and not only symmetric A), it must be over the complex plane.

Nonetheless, there appears to be a general trend among recently proposed (recurrence based) sequence primitives such as the s6 SSP[32], the recurrent sequence primitive used in RG-LRU[20] and the recurrent sequence primitive used in HGRN2[70] to use diagonal but non complex valued transition matrices. In all 3 instances the respective authors observe that real valued state transition matrices perform better on NLP tasks, but argues that complex parametrizations potentially remain relevant for other modalities.

On a final note, it is observed that these 3 non-complex-valued instances all employ input dependent gating mechanisms whereas their 3 respective precursors did not [36],[62],[69]. One speculation proposed in this thesis is that the importance/necessity of complex parameterization is not related to the modality of the data, but rather whether the recurrent sequence primitives are input dependent.

10 Introduction to the Experimental Work

From here on the thesis will be almost entirely practical. The overarching focus is accessing the performance of the 3 introduced SSPs s4, s4d and s6 in a variety of settings and circumstances. Moreover numerous attempts will be employed in the pursuit of enhancing or improving the baseline SSPs. Many of these attempts are related to the subject of the inherent directionality of the SSP based models. Each of these extensions will be covered in detail and be thoroughly empirically evaluated.

10.1 Scope of the Experiments

The focus will not be their linear scaling properties. A great advocate for SSPs based deep models lies in their linear or near linear scaling in sequence length. The theoretical part of the thesis often highlighted the scaling properties, and the computation and memory efficiency of the SSPs. Moreover multiple comparisons between SSPs and other sequence primates have been drawn to accentuate advantageous properties of the SSPs. Evidently these properties have already had a great deal of coverage in the theoretical part of the thesis namely because they are theoretically grounded. As such rather than further supporting these claims regarding through experimental validation they are accepted at face value and will only play a minor part of the practical section of this thesis.

The focus will not be a comparison to non-SSP based models. Empirical comparison to traditional sequence models such as transformers, RNNs and all their variants are also of less interest. Comparisons to traditional methods will emerge only in the form of forwarding results reported by their respective papers, rather than attempting to replicate their results by rerunning their benchmarks. It will be clearly stated whether results are merely forwarded and results will only be forwarded in instances where all circumstances are similar to ensure a fair comparison.

The focus is intra-SSP modeling capabilities. The focus of this practical section will be an "intra-SSP" exploration of the introduced SSPs with the primary goal being to quantify their modeling expressivity and capabilities. As stated comparisons will mainly be with respect to different SSPs, rather than different general sequence primitives (RNNs-recurrences, self-attention, etc). In the pursuit of investigating expressivity, multiple techniques which have proved crucial for expressivity in other deep learning fields, will be employed in the context of the SSPs.

The experiments are not constrained to SSP based models. While all experiments involve and are conducted using deep SSP based models, it is noted that the subjects investigated in the experimental sections is some what agnostic to these models. The experimental subjects are only directly tied to the inherent directionality of the deep model and as such may generalize to any deep directional model and in particular deep recurrent models. For instance all experiments could be rerun using the Linear Recurrent Unit[62] as the sequence primitive of the deep model rather than s4, s4d and s6. Obviously the results would be vastly different and the discussions and conclusions would not apply.

10.2 Topics of the Experimental Sections

The experimental part of the thesis will be split into 6 sections:

1. **SSPs in deep sequence models**: A section containing all the practical details such as 1) The different network architectures and how they are equipped with SSPs. 2) Considerations in regard to facilitating a fair comparison with respect to model size and compute. 3) Optimization, initialization and hyperparameters details.
2. **Introducing the Long Range Arena**: LRA is a benchmark suite consisting of 5 different tasks specifically designed for long range sequence modeling. It has become widely adopted in the long range sequence modeling field and LRA results consistently appear whenever a new model is proposed. The

LRA tasks will serve to facilitate most experimental work conducted in this thesis and thus needs a thorough introduction.

3. **Establishing a baseline**: That is, reproducing central results reported by the s4 and s4d paper. First and foremost this will serve as a sanity check to confirm that everything is in order before introducing modifications to the SSPs. Moreover, the baseline experiments will also serve as the opportunity to control for the effect of different choices of network architectures. Finally this section touches upon categorical versus numerical representation of the same data and the potential impact hereof.
4. **Investigating the effect of positional embeddings:** Including some form of positional encoding has proved crucial to achieving optimal performance in transformers. The need for positional encoding in the self-attention mechanism has an intuitive explanation. Similarly intuitions exist in regard to why SSPs do **not** need positional encoding. Nonetheless theory and practice do not always align and (to my knowledge) the effect of positional encoding in SSP based models has not been empirically quantified.
5. **Investigating the effect of bidirectionality**: SSPs are inherently both causal and directional in the manner they process an input. Some problems however, do not require a causal model and some modalities are only directional by convention. Consequently it may prove beneficial to equip the SSPs with bidirectionality when applying them to tasks which allow it.
6. **Exploring approaches to pretraining bidirectional models**: The final experimental section relies on 2 assumptions. Unsupervised pretraining improves performance on downstream tasks. Moreover it is assumed that equipping the SSPs based models with bidirectionality improves performance on suitable tasks. Under these conditions it is of interest to investigate the impact of MLM pretraining of a bidirectional model versus NTP pretraining of a unidirectional model. Lastly a hybrid approach is attempted which constitutes in pretraining a causal model and subsequently morphing it into a bidirectional model for downstream tasks. The idea being to leverage potentially more potent causal pretraining while still enabling bidirectionality for downstream tasks.

10.3 Code

All code can be found by follow the link below:

`https://github.com/KarlUlbaek/Deep-Learning-with-State-Spaces-for-Efficient-Sequence-Modeling`

11 The Long Range Arena Tasks

This section introduces the sequence modeling tasks used to facilitate the majority of the experimental work conducted in this thesis.

11.1 Introducing The Long Range Arena (LRA)

The widely accepted default suite of benchmark problems within the efficient sequence modeling literature is known as LRA. LRA is to efficient sequence modeling what imagenet[21] and CoCo[50] is to the computer vision literature. LRA stands for Long Range Arena and originates from the paper[78]. LRA consists of 5 problems which are specifically designed to require the model to be able to model long range dependencies. To make the benchmark as widely applicable as possible the problems deliberately do not require a causal model. Moreover the problems are relatively small scale as to not require industry grade hardware. Furthermore the small scale aligns with the author's intention/request to not use pretrained models. All problems are classification problems, but vary in their modality and classification objective. The 5 problems are outlined as follows:

- **Character level sentiment analysis**: This task uses the IMDB review dataset and poses a binary classification problem. Given a text review, did the reviewer like or dislike the movie. The review is presented to the model on a character-level (sometimes denoted byte-level) rather than the usual word level. Input sequences are at most 4000 characters long. Shorter reviews are zero padded while long reviews are cut off.
- **Character level document retrieval**: A binary classification task, where the goal is to answer whether 2 scientific papers are related. Two papers are considered related if either cites the other. The sequence fed to the model consists of the first 4000 characters of one paper followed by the first 4000 characters of the other paper for a total sequence length of 8000.
- **Long List operations**: A synthetically generated task that is best described via an example. Each sequence is a nested list of operations and could look as follows:
  `MAX 4 3 [MIN 2 3 ] 1 0 [MEDIAN 1 5 8 9, 2]]` The goal is to determine what performing the list operations will amount to. The operators min, max etc er picked to ensure that the result remains in the range 0-9, resulting in a 10-way classification problem. In the above example the correct answer is 5. The data is synthetically generated in such a way, that the length of the sequences varies in length in the range 2k-16k elements. All sequences are cut off at 2k elements.
- **Image classification on sequential Cifar10**: This task uses the Cifar10 image classification dataset. All images are converted to grayscale and flattened resulting in each image being a sequence of 1024×1. The grayscaled pixel values are rounded to the nearest whole number and treated as categorical input with 256 possible categories. Since the embedding layer is initialized at random, the pixels are stripped of their relative meaning: In the eyes of the embedding layer a pixel from category 2 is not closer related to the pixel from category 3 than it is to the pixel from category 256. The fact that the pixel values become meaningless (until relationships/patterns are learned) makes the problem particularly difficult and becomes of intricate importance momentarily.
- **Pathfinder**: A synthetic binary image classification task best understood through inspection of the 2 example images seen in figure 8 below. The problem consists of determining whether there exists a dashed path/line between the two dots. The images are inherently grayscale. Similar to the Cifar10 setting the images are flattened and the pixel intensities are rounded off and treated as categorical inputs. Equivalent reservations applies in regard to how the pixels are stripped of their relative meaning as a result of the categorical formulation. The default image resolution is 32x32 implying a sequence length of 1024. Path-X denotes the most challenging setting in which the images are of resolution 128×128 resulting in sequences of length 16384.

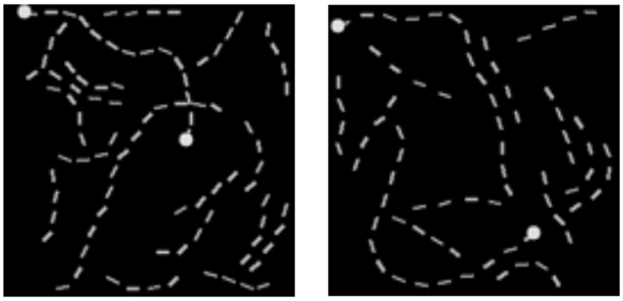

Figure 8: Pathfinder example: The left image is positive sample since there is a dashed line connection the 2 dots while the right image is a negative sample as no line connects the dots.

11.1.1 Omitting Certain Variants of LRA

While the 5 problems are small scale toy problems, sequence modeling is a computationally expensive endeavor and the training runs take a substantial amount of time even on high end hardware. Therefore certain variants of the LRA task are in this thesis omitted with the following reasoning:

The character Level document trivial task was omitted as it was deemed too similar to the character sentiment analysis (IMDB-review). Both task are character-level binary text classification tasks, with the character Level document trivial task being twice the sequence length implying twice the training time.

Furthermore the long list operation task was omitted as it is fundamentally flawed. The list sequences are cut off at 2000 elements, while the full sequence ranges upwards of 16000 elements. The abrupt cutoff ruins the careful nested structure of the data. The flawed nature is supported by the fact that SOTA models are only able to achieve around 60% accuracy on this task compared to the 90+% accuracy on the other 4 tasks. It would have been interesting to empirically support the claim that the task is flawed. It could be experimentally demonstrated by changing the sequence length to 16000 such that no cutoff occurs, but is beyond the scope and interest of this thesis.

Lastly Path-x (16k sequence length Pathfinder) is omitted. When the LRA paper proposed Path-x it was intended as the most difficult problem. The result section of the s4 paper however, demonstrates that the regular Pathfinder (1024 sequence length) is more challenging than Path-x. The implications being that Path-x is an easier task which requires 16 times the training time, deeming it redundant.

11.1.2 Expanding upon LRA

While 2 of the 5 tasks (not including Path-x since Pathfinder remains) are actively omitted for the above stated reasons, 2 additional tasks are included instead. The 2 additional tasks are not entirely new but rather substantial alterations of Pathfinder and sequential Cifar10. The motivation to include the alterations are a result of a particular finding in the s4 code repository. From their experiment configuration files it is evident their results are produced using the numerical valued pixel intensities rather than rounding them off and assigning them to categories. Moreover in the setting of Cifar10 it appears the images are not grayscaled and instead remain RGB. Both these initiatives are hypothesized to ease the classification task: the numerical pixel values possess relative information, and the RGB values even more so than their grayscale counterparts.

The different versions of Pathfinder and Cifar10 will be suffixed "categorical" or referenced as the categorical version to indicate that the pixels are treated as categorical inputs as intended by LRA. Correspondingly numerical, i.e. Cifar10-numerical, denotes that the pixel values remain floating point numbers and even RGB in the Cifar10 instance. Recall that the numerical valued input sequences omits the one hot encoding and are instead directly multiplied with the linear embedding projection layer. Moreover Cifar10 remaining RGB implies the input sequences are of dimension 1024x3.

11.1.3 SSP Based Models Were the First to "solve" LRA.

At the time of LRAs original publication in 2020 no available models were able to solve the LRA problems to a satisfactory degree. When s4 was published it beat the previous LRA SOTA model by more than 20 accuray points. Consequently s4 represents a great milestone in the efficient sequence modeling literature and solidifies the capabilities of state space based models.

12 Incorporating SSPs into a Deep Sequence Model

Up until this point this thesis has only discussed the heart of the deep state space model notably the s4, s4d and s6 variants. While these are what ultimately determine the properties of the sequence model they only constitute a small part of the whole deep state space model architecture. Their character defining properties lie in the fact that they are responsible for temporal mixing which is mixing along the sequence dimension. Recall that s4 and s4d treats/models the different features of the token embeddings entirely independently. Consequently, the equally important but not equally difficult, job of feature mixing (mixing along the token embedding dimension) is required to happen elsewhere in the network.

12.1 Two Deep Neural Network Architectures

The recipe for a generic deep sequence model was introduced in section 4.2.3. The concrete two deep sequence architecture used in the s4/s4d (they share architecture) and s6 paper will now be covered in full. Both architectures employ a simple embedding module and a simple decoder module in accordance with the description found in section 4.1.2. Notice that the LRA tasks are all classifications task thus `mean-pooling` is applied before the decoder module.

Where the 2 architectures differ are in the structure of the K repeated network blocks. The s6 paper employs a vastly more sophisticated approach to the composition of its block. In general the s6 paper puts great emphasis on their proposed block and considers it one of the noteworthy contributions of the paper. They term their proposed deep sequence architecture Mamba and dub the blocks for Mamba-blocks. On the contrary the s4/s4d block is unnamed in the 2 papers and *simpler* compared to the Mamba-block. Therefore it will in this thesis be denoted the "Simple block". A figure illustrating the 2 different network blocks can be seen below followed by notes explaining each block.

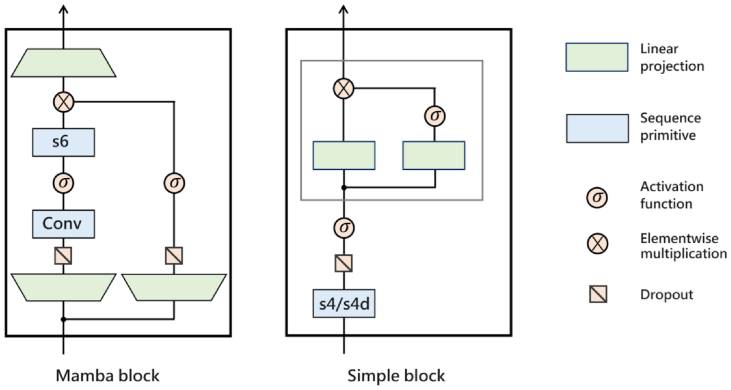

Figure 9: Illustration of the 2 different deep network blocks. The contents of the gray box within the Simple-block is commonly referred to as a gated linear unit(GLU).

The Mamba block:

- The 2 initial linear projections of the Mamba block expands the token embedding dimension 2 fold and correspondingly the output projection halves the token embedding dimension.
- The "Conv"-module falls into the definition of being a sequence primitive although a **very simple** sequence primitive. "Conv" are causal 1d convolutions with kernel length 4 and a bias term. There are d convolution kernels/filters, that is one for each feature in the token embedding. The convolutions are applied along the sequence and are depth-wise-separable. i.e. all features are treated independent and no feature mixing occurs.
- Both activations in the Mamba block are SiLU activation functions.

The Simple block:

- The activation succeeding the s4/s4d layer is a GELU activation function.
- The activation succeeding the linear projection on the right branch is the sigmoid function.
- The particular processing occurring in the inner gray box of the "Simple block" is commonly referred to as a gated linear unit or GLU for short.

12.1.1 Fused Mamba-block

The Mamba block source implementation comes in two flavors, one optimized for speed and memory efficiency one which compromises speed to provide modularity. The one optimized for speed will be denoted be denoted "fused-Mamba-s6" in this thesis. Recall the concept of recomputation explained in section8.2. The "fused-Mamba-s6" block similarly applies recomputation but on a block level. Concretely they discard certain "fast to recompute" intermediate activations, such as those produced by activation functions and 1d conv layer. The fused-Mamba-block is only available with the s6 SSP. Note however that any model based on the Mamba block is applicable for a similar fused treatment, but would require the time and effort associated with the customized low level implementation.

From a practical perspective the implications are that in the fused variant almost the entire block is abstracted away to a single c++ function call, making the Mamba block architecture difficult to modify as a practitioner. Fortunately the source code also includes a non-fused, variant. The non-fused variant is almost purely written in PyTorch, thus making modification and experimentation easy. The performance implications of the fused versus non-fused variant is demonstrated in later sections.

12.1.2 Residual Connection and Normalization

Both block types are wrapped in a residual connection (not visible in the figure). Moreover both block types employ some form of normalization. In the instance of the Simple block the specific type of and location of the normalization layer may either be pre- or post-block and vary from layer to batch normalization. In the s4 paper these two choices are treated as tweakable hyperparameters and tuned according to the specific problem. For all experimental work conducted in this thesis all models based on the Simple block are consistently equipped with post-block layer norm.

The Mamba architecture as presented in the s6 paper exclusively employs root mean squared (RMS) normalization[88] in a pre-block manner. This particular choice is motivated from a performance and efficiency standpoint. Compared to layer norm, RMS norm is slightly faster to compute. Furthermore the pre-block location allows for fusing the norm and residual computation, further optimizing the speed of the architecture.

12.1.3 Dropout and Weight Decay

Both blocks use 1d feature-wise dropout. To be explicit, when some feature index of the token embedding is randomly selected for dropout, said feature will be disabled (zeroed out) for all token embeddings in the whole sequence.

The source implementation of the s4 and s4d SSPs includes dropout applied directly to the state space kernel $\bar{K}$, right before it is Fourier transformed and convolved with the input sequence. Their experimental configuration files suggest that this dropout is primarily disabled, which will exclusively be the case in this thesis.

The source implementation of the Mamba block does not include any dropout. The s6 paper does not address the reason directly. One possible explanation arises from the fact that s6 and the Mamba architecture is framed (by the authors) as an attempt to improve upon s4 based models in regard to categorical modalities, particularly natural language. The SOTA transformer models they benchmark themselves against, and thus presumably inspired by, are PaLM[30] and LLaMA2[79]. These two models do not utilize dropout, but do however employ aggressive weight decay. In the NLP literature it is common to not to employ dropout as the (pre) training corpora are so large that training only last for 1 epoch.

12.1.4 The Mamba-block Requires Regularizing

The suite of baseline benchmarks presented momentarily consists of problems which require training for a large amount of epochs. Pilot experiments conducted in this thesis and seen seen in figure 10 displays that in these instances the vanilla Mamba architecture will suffer when it is not equipped with dropout. The amount of dropout appears secondary as long as some is present.

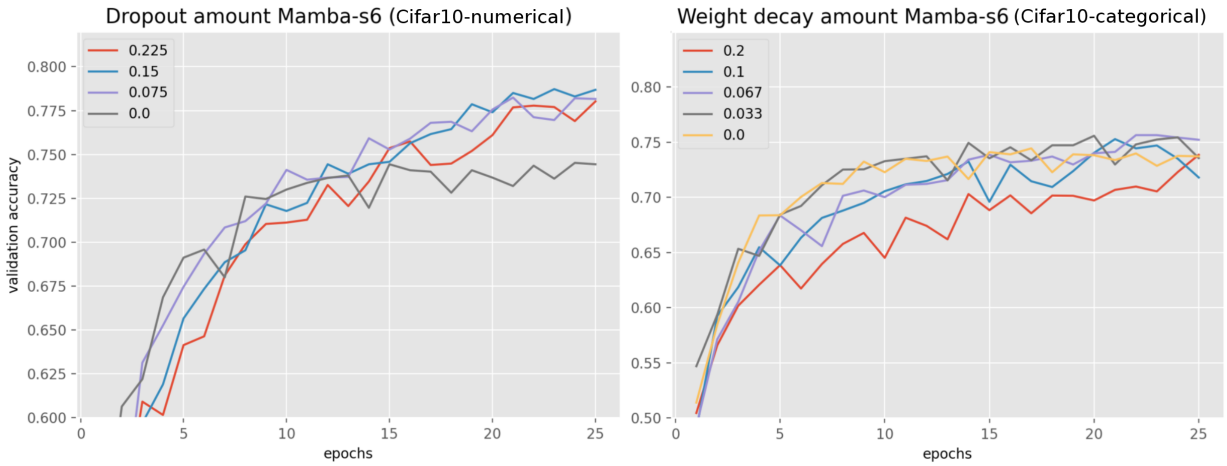

Figure 10: Mamba-s6 dropout and weight decay pilot experiment using one of the baseline tasks from the LRA benchmark suit presented in next section.

Inspired by the aggressive weight decay used by PaLM and LLaMA, further pilot experiments were conducted to investigate if weight decay could provide a similar regularizing effect as dropout. As seen in figure 10 weight decay does not provide a boost to the final test accuracy as it was the case with dropout. The only noticeable effect appears to be that, the greater the weight decay the longer it takes for the model to converge. Consequently weight decay was left at the default PyTorch AdamW value of 0.01 for all subsequent experiments and models.

A note on the location within the Mamba block chosen for dropout: For consistency the obvious choice would be to match that of the Simple block, i.e. to place dropout right after the s6 module. This approach however is not directly applicable due to the fused-Mamba-s6 variant described in section 12.1.1. The chosen dropout location indicated in figure 9 reflects compatibility with the fused-Mamba-s6 variant while being as close to the SSP as possible.

12.2 Model Size and Model Hyperparameters

The experimental work conducted in this thesis has a very minimalistic approach to hyperparameter tuning. The modified version of LRA contains 5 tasks and a total of 5 different models will be assessed resulting in a total of to 25 training runs. Hyperparameter tuning each model and task would rapidly grow out of proportions. Consequently next to no hyperparameter tuning has been conducted, with learning rate being the only exception. It was tweaked on a per task basis by simply decreasing it until training appeared stable across all models.

12.2.1 The Baseline Hyperparameters

All remaining hyperparameter were inspired by the configurations used in the s4 paper. The s4 paper includes 2 sets of hyperparameter configurations as a consequence of the authors rerunning the experiments and updating the results after the paper had been published. The configuration used in this thesis is a hybrid approach between the two.

	Layers	(Mamba) d & N	(Simple) d & N	LR	Dropout	Epochs
IMDB	6	116 & 16	170 & 64	0.001	0.1	35
Cifar10*	-	-	-	0.003	0.15	100
PathX*	-	-	-	0.0001	0.05	50

Table 2: Default hyperparameters used for all experiments conducted in this thesis unless otherwise stated. N is the state dimension and d is the token embedding dimension. The Cifar10* and PathX* parameters are shared across their respective categorical and numerical variants.

12.2.2 Tying Parameter Count Across Models and Experiments

The Simple block and the Mamba block have state dimension N of size 64 and 16 respectively, which are the default and only value used in their corresponding papers. The token embedding dimension d is chosen such that all combinations of block and SSPs have trainable parameter count close to 600k (in practice 550k-650k depending on the specific SSP/block combination and input and output dimension of the task, see table 2). Trainable parameter is deemed the principle quantity to keep fixed in order to facilitate fair comparison between the different models. All models throughout all experimental sections have trainable parameter count in the 600k range. FLOPs would have been an equally good choice although a more complicated one. For instance during the backward pass s6 uses recomputation to save memory. Moreover s4 and s4d uses relatively less compute when the batch size is increased as the global kernel only needs to be constructed once (per layer) per batch. Those are 2 instances where using FLOPs as the measure would complicate matters and as such trainable parameters were picked due to its simplicity.

12.2.3 Effective State Size and Parameter Distribution

One detail which may appear conspicuous is the fact the Mamba block both has a smaller model and hidden state dimension. Recall that the linear input projection of the Mamba block has an expansion factor of 2. This implies that the when the data is processed by the SSP, the data has token embedding dimension $d = 2 \cdot 116 = 232$. In this sense the SSPs in the Mamba block are of higher dimension than those contained in the Simple block. While there are more SISO-channels in the Mamba block the state dimension N may too be considered an expansion factor. With this perspective the Mamba block has combined state dimension of $d \cdot N = 232 \cdot 16 = 3712$ compared to the $d \cdot N = 170 \cdot 64 = 10880$ of the Simple block. Notice these two numbers essentially represent the raw amount of memory in terms of floating point numbers available for representing the context of the whole sequence. The combined dimensionality of the Simple block is considerably larger implying a large memory budget to store context of the sequence. Moreover the Simple block generally has a substantially greater amount of its parameters present as state space parameters as indicated by the following table.

	Mamba-s6	Mamba-s4d	Mamba-s4	Simple-s4d	Simple-s4
Full model	587k	564k	587k	551k	616k
in SSP	95k	72k	95k	200k	266k
% in SSP	16%	13%	16%	36%	43%

Table 3: Number of trainable hyperparameters in the entire model and within the SSPs of the model. The exact values are taken from the Cifar10-numerical task. Slight variances will occur when the task is changed due to the change in problem specific embedding and decoding dimension.

In regard to table 3: in all instances the "% in SSP" will only further decrease if the model is scaled as a result of the relative scaling in d of linear projections layers compared to the state space layers. Although being a sequence primitive the table does not consider the 1d conv layer of the Mamba block as part of the SSP parameters. Across all 6 blocks the 1d conv layer constitutes a diminishingly small 7k parameters.

One noteworthy observation to be had from table 3 is that the Mamba block only posses $\sim$ 15% of its parameters in the SSP module whereas in the Simple block the SSP module constitutes $\sim$ 40% of the parameters. Recall that the SSPs are solely responsible for mixing along the channel dimension whereas the remaining parameters (linear projections) exclusively mix across the sequence dimension. This gives an indication that the Mamba block emphasises feature mixing while the Simple block prioritizes temporal mixing.

12.2.4 Initialization, Optimizer and Stability.

For all experiments the model parameters are initialized as follows:

- All B are initialized according to the LegS initialization from equation(76) which looks as follows: $B_n = (2n+1)^{\frac{1}{2}}$
- Recall that the HiPPO-LegS A_{LegS} matrix was not employed directly but instead represented as the DPLR decomposition: $A_{LegS} \approx \Lambda - QQ^\top$
- In s4 the A is initialized according to the full DPLR decomposition.
- In s4d the A is initialized as Λ from the DPLR decomposition.
- In s6 the A matrix is initialized as $\mathbb{R}(\Lambda)$ from the DPLR decomposition.
- The Δ parameters are sampled uniformly from (0.001, 0.1).
- The C parameters are drawn from a standard Gaussian and does not undergo rescaling.
- The D parameters essentially being skip-connections are initialized to all ones.
- In the instance of s6, (B, C, Δ) does not follow the above initialization as they materialize from input dependant functions (primarily linear projections).
- All linear projection layers use default PyTorch initialization.
- The 1d conv layer in the Mamba block uses default PyTorch initialization.

All training uses a batch size of 64 and the AdamW optimizer, with default parameters here under a weight decay of 0.01. Learning rates are superficially tuned for each problem to the largest value that yields a stable training. Cosine Decay is used for learning rate scheduling with a minimum value of 0.1 of the initial learning rate. To enforce stability the s4 paper notes that one should exhibit caution with regard to the SSP parameters and in particular A. To accommodate the stability of the sensitive SSP parameters (A, B, C, Δ) during training their learning rate are always scaled to be 0.1 of the learning rate used for the rest of the parameters. Furthermore weight decay is always disabled for (A, B, C, Δ). No experiments were conducted to investigate if these precautions actually promoted stability. In the setting of s6 the aforementioned precautions only apply to A.

12.2.5 Methodology for Assessing Computational Overhead

In section 12.2.2 it was established that to enforce a fair comparison between the models the number of trainable parameters would always be fixed to $\sim$600k. Nonetheless certain modifications applied in the subsequent experimental sections may induce a some amount of computational and memory overhead even though the amount of parameters have been controlled for. This short section covers the methodology used to asses any potential computational overhead.

Approach. Computational overhead is assessed by averaging the time it takes to process 200 batches of size 64. The 64 elements in the batch are sequences of size 1024×3. This corresponds to the exact batch size and sequence dimension of the Cifar10-numerical task. In order to circumvent potential dataloader overhead purely synthetic data is used instead of the Cifar10 data. The test is conducted using the 600k parameter models found in table 3. Memory is assessed as the peak memory usage during the aforementioned synthetic batch processing. The experiment is conducted using an A100 80GB GPU.

Reservations. The methodology to assess the computational overhead is subject to a number of reservations. For instance it was not investigated how the relative performance gap scales with model size or sequence length. Moreover, the results are purely empirical and in terms of concrete wall-clock-times. As a result they depend on the specific code implementation and concrete hardware optimization. The alternative (neglected) approach would have been to theoretically determine the exact amount of floating point operations and memory usage in either instance. This approach however, is unable to account for memory bandwidth constraints. These are often the true limiting factor in regard to data processing speed as highlighted by the Flash Attention paper[19] and the s6 paper itself. Ideally both theoretical and empirical analysis should be conducted. Theoretical FLOP analysis was deemed too time consuming given that computational overhead analysis only constitutes a minor side-result in this thesis.

13 Establishing Basline Results on LRA

The primary purpose of the LRA baselines benchmarks is 2 fold. It serves as a sanity check to ensure the s4 and s4d models are performant i.e. that the results reported in their respective papers can be reproduced to a satisfactory degree. More importantly the baseline LRA section is used to facilitate experiments which seek to investigate the effect of the SSPs versus the network architectures. In practice this amounts to exchanging the s6 module in the Mamba block with either an s4 or s4d module. In total 5 different models will be tested on LRA: Simple-s4, Simple-s4d, Mamba-s4, Mamba s4d and Mamba-s6. (The Simple-s6 variant was deemed redundant. The notion being the Mamba architecture would not have been developed for s6 in the first place if it did not improve performance over previous architectures.)

Remarkably the s6 paper does not provide LRA benchmark results for their model, while all previous publications from the same author consistently include them. One possible explanation is that s6 and Mamba are designed and meant for large scale NLP modeling and LRA is deemed uninteresting. A more probable explanation is that s6 performs underwhelmingly on the LRA benchmarks, which in itself is noteworthy considering its impressive performance in other aspects. One complimentary contribution by this thesis is LRA results for s6 and Mamba.

13.1 The LRA Results

Below is a table containing the test accuracies from the 5 different models trained in this work. The table additionally contains results from the s4 and s4d paper as well as the SOTA model. Details related to the different models are elaborated in full under the table.

Model / Task	IMDB	Cifar10 (num)	Cifar10 (cat)	Pathfinder (num)	Pathfinder (cat)	Average
s4(old)†	76.0	87.3		86.1		83.1
s4(new)†	86.8	88.7		94.2		89.9
s4d†	86.2	88.2		93.1		89.2
Mega†	90.4		90.4		96.0	92.3
Mega-chunk†	90.2		85.8		94.4	90.1
Simple-s4d	75.6	84.5	73.9	91.8	97.0	85.7
Simple-s4	79.2	87.0	72.4	86.2	97.2	87.8
Mamba-s6	89.6	81.4	69.8	74.0	93.5	88.2
Mamba-s4d	88.9	86.3	78.1	95.8	96.9	90.7
Mamba-s4	89.9	83.8	75.3	85.6	95.5	89.7

Table 4: Test performance on the LRA benchmark suite. Values in gray were omitted from the avg calculations for reasons explained below. All models marked with † are reference models and their accuracies are forwarded from their respective papers. The s4† (old), s4† (new) and s4d† uses the numerical variants of Cifar10 and Pathfinder while Mega† and Mega-chunk† uses the categorical variants.

Details Related to the Reference Models†.

- All results regarding the reference models were not reproduced but merely restated from their respective papers.
- Mega[54] holds the SOTA result on the LRA benchmark suite. Mega uses dense self-attention interweaved with exponential decay to effectively model long sequences. The use of dense self-attention implies quadratic scaling in sequence length. Mega-chunk employs the same fundamental approach as Mega but using local attention/sliding window attention resulting in linear scaling in sequence length.
- From the code repository belonging to Mega it appears they use the categorical implementation of Pathfinder and Cifar10 as intended by LRA.

- All reference models† (except all s4(old) and s4(new)-IMDB) have substantially more than 600k trainable parameters. Additionally most reference models† were trained for a greater amount of epochs than used in this work. For the exact values consult the respective papers.
- The s4(new)† and s4d† models are based on the Simple block but are typically much larger and bidirectional.

Details and Observations Regarding Models Trained in this Work.

- As suspected there is a consistent and considerable discrepancy between the numerical and the categorical version of Cifar10 across all 5 tested models. It remains unknown whether the authors of s4 and s4d were aware of this and actively choose the more performant task for their results. Neither is it known whether Mega too would be more performant on the numerical version or if attention based approaches have a preference for categorical data. This is an interesting subject for future research and may reveal differences between attention based sequence primitives and SSPs.
- On the contrary and to some surprise the performance relationship between categorical and numerical is reversed in the setting of the Pathfinder task. In this instance all 5 models perform substantially worse in the numerical setting. Before jumping to any conclusions, it is suspected that the Pathfinder numerical implementation may be faulty or contain some vital oversight. This suspicion arises from inspection of figure 11 below which illustrates the learning process of the models in the two Pathfinder settings. It is immediately evident that in the numerical setting all models struggle to find a foothold, taking respectively 1, 5, 6, 10 and 17 epochs before they learn meaningful patterns and arise above random guessing. In many instances the models never learned (not included in the figure). This particular behavior was specific to the Pathfinder numerical setting, as learning would consistently occur in the 0th epoch in all other tasks. Similar learning plots for the rest of the tasks can be found in appendix17.
- The cause of the problem was never uncovered. In particular a variety of learning rates were attempted but to no avail. It was ensured that the data was properly normalized and in the range -1 to 1. In addition other normalization schemes were attempted but were not found to remedy the learning issue.
- With the benefits of hindsight, the issue is thought to be related to the initialization of the models and in particular initialization of the Discretization parameter Δ. It appears the s4 and s4d paper uses a different more carefully initializing scheme for Δ in the Pathfinder task. This hypothesis was never tested.
- In the spirit of the s4 and s4d paper only the superior Cifar10-numerical results are considered. Furthermore only the Pathfinder-categorical results are regarded, due to the mysteriously impeded learning process of its numerical counterpart.

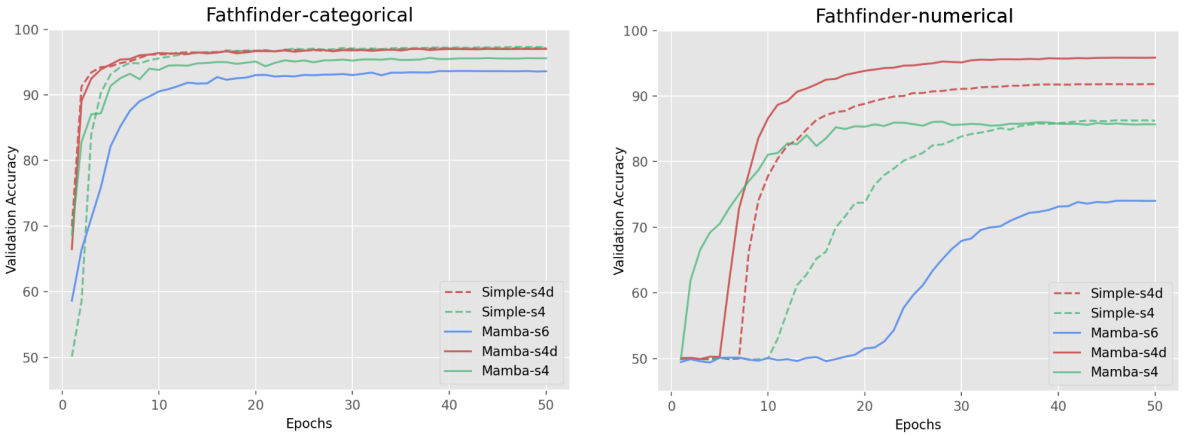

Figure 11: Validation learning curves of the two Pathfinder versions. The Pathfinder-numerical results strongly indicate an inherent flaw.

13.2 Findings

It was not the goal to produce new SOTA reults on LRA but instead to ensure that the implementations in this work perform as they should. Disregarding the Pathfinder results it appears that all models trained in this work are working and performing as expected.

Moreover it was sought to investigate which of the Mamba and Simple block were more performant. In this regard Mamba appears as the superior architecture. Given the parameter distributing seen in table 3, Mamba emerging superior may suggests that only a small amount of parameters needs to be allocated to the sequence primitives and emphasis should be feature mixing, rather than temporal mixing.

As expected s4 and s4d perform close to identical rendering the additional complexity of s4 redundant. Their throughput and memory usage are also close to identical. s6 falls slightly behind s4 and and s4d, which may explain why the s6 paper does not include LRA results. Nonetheless the s6 paper acknowledges that s6 may be unfit for certain tasks as they themselves include certain audio benchmarks in which they too fall behind s4 and s4d. s6 remains competitive on the IMDB task. These observations suggest that s6 has a preference for inherent categorical modalities while being less performant on inherent continuous/numerical modalities such as image and audio.

In particular this works Mamba based models (Mamba-s6, Mamba-s4, Mamba-s4d) are highly competitive with the SOTA model Mega[†]. Even more so when considering that (Mamba-s6, Mamba-s4, Mamba-s4d) for the most part have less parameters, are trained for less epochs, have undergone less task specific hyperparameter tuning and are non-bidirectional. While this was never of interested, it is noted that accommodating the aforementioned reservation would potentially allow for new SOTA LRA results.

14 Investigating the Effect of Positional Embeddings:

This section is dedicated to experimental investigation of whether the addition of the positional embeddings can improve performance of the SSP based models.

14.1 Motivation and Background

The motivation for the investigation is twofold. Positional embeddings have proved crucial for the success of the attention mechanism and when positional embeddings improved the transformer architecture as a whole improved. Moreover, (to the best of my knowledge) there is no experimental evidence in the literature of deep state space based models that either confirms or rejects the potential benefit of positional embeddings. The concept does not even appear to be mentioned anywhere. The complete absence of the concept in the literature may itself be a testament to the fact that it is entirely redundant: that the inherent sequential and causal nature of the recurrent formulation implicitly asserts a distance relationship between the tokens in the sequence. Nonetheless, the field of deep learning does not always comply with intuition and concepts should always be experimentally proven or disproven.

Positional Embeddings in the Context of Self-Attention. Consider positional embeddings in the context of the attention mechanism. When the transformer was introduced in 2017[80] the authors acknowledged the need for positional encoding. The need stems from the fact that the attention mechanism is permutation equivariant, that is invariant to the order of the tokens: if you were to swap the location of 2 tokens in the sequence the attention-value between all tokens remains the same their location in the attention matrix has just moved.

To accommodate the *position agnostic* nature of self-attention 2 forms of positional encoding were proposed. Both proposals are instances of absolute positional encoding (APE) and involve adding (learnable or fixed sinusoidal) embedding information onto each token. In either instance the information is a function of the absolute token index in the sequences. Since then, a whole body of research has gone into improving the positional encoding used in transformers. Many of these improved proposals focus instead on a relative positional embedding (RPE) approach. Rather than having the positional information be a function of the absolute token index in the sequence, they are instead a function of the relative index distance between two tokens. Many popular RPE methods[17][67][15][16] can be described by the generic formula[48]:

$$RPE_{\text{generic}}\left(X\right) = \mathbb{A}_{PRE} + R, \qquad \mathbb{A}_{PRE} = XW^{(Q)}\left(XW^{(K)}\right)^{T} \tag{78}$$

Where W are linear projections producing the queries and keys, $\mathbb{A}_{PRE}$ is the $L \times L$ "presoftmax-attention-matrix" (see equation 7 for the complete attention matrix) and R is an additive $L \times L$ matrix induced by some parameterized function. The specifics of the parameterization is where the various methods differentiate themselves from one another. In all instances R serves to penalize the attention-value between 2 tokens proportional to their relative distance in the sequences. Evidently, these approaches rely on injecting the RPE information directly into the attention-matrix, making them inapplicable for anything but attention based sequence primitives.

14.2 Rotary Positional Embeddings (RoPE)

RoPE[76] is a particularly popular RPE approach used in some of the best open source LLMs such as PaLM[30], LlaMA2[79] and Mixtral[44]. RoPE stands for rotary positional embeddings and is the embedding approach which will be employed and experimentally investigated in this thesis. The choice to employ RoPE is 2 fold. It being the preferred choice for open source SOTA LLMs validates its effectiveness. More importantly, although RoPE is designed with the attention mechanism in mind it is not subject to the generic formulation in equation (78). Following the notation used in the previous equation RoPE emerge as:

$$RPE_{\text{RoPE}}(X) = \mathbb{A}_{PRE}, \qquad \mathbb{A}_{PRE} = \text{RoPE}\left(XW^{(Q)}\right)\text{RoPE}\left(XW^{(K)}\right)^T \tag{79}$$

Evidently the embedding information is applied to the queries and the keys prior to their outer product being computed. Although originally intended to be applied to queries and keys RoPE is applicable to any sequence X. RoPE encodes the tokens with relative positional information by rotating them with an angle proportional to their index in the sequence. As such, RoPE entails that the further 2 tokens are from each other in the sequence the greater the relative angle between them. The reason to use relative rotation to encode relative distance is again tied to self-attention and explains why RoPE is applied to both the keys and the queries. The similarity measure used in attention is the dot/inner product. The authors of the RoPE paper are able to analytically show that the dot product i.e. attention decays when the angle is increased (see figure 2 in the RoPE paper).

In practice RoPE is applied to the k'th token embedding through matrix multiplication with a particular $d \times d$ rotation matrix $R^d_{\Theta,k}$.

$$\text{RoPE}(x_k) = R^d_{\theta,k} x_k \tag{80}$$

The rotation matrix emerge as the generalization of the usual 2-dimensional rotation matrix repeated along the diagonal $d/2$ times:

$$R^d_{\theta,k} = \begin{pmatrix} \cos k\theta_1 & -\sin k\theta_1 & 0 & 0 & \cdots & 0 & 0 \\ \sin k\theta_1 & \cos k\theta_1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \cos k\theta_2 & -\sin k\theta_2 & \cdots & 0 & 0 \\ 0 & 0 & \sin k\theta_2 & \cos k\theta_2 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & \cos k\theta_{d/2} & -\sin k\theta_{d/2} \\ 0 & 0 & 0 & 0 & \cdots & \sin k\theta_{d/2} & \cos k\theta_{d/2} \end{pmatrix} \tag{81}$$

θ is a predetermined parameter inspired by the fixed sinusoidal APE proposed in the original transformer paper and given by:

$$\theta = \left\{\theta_i = 10000^{-2(k-1)/d}, \quad k \in [1, 2, \ldots, d/2]\right\} \tag{82}$$

Notice that the rotation matrix depends on the current index k. Materializing the L different rotation matrices and performing L matrix vector products $R^d_{\theta,k}x_k, \quad k \in \{0..L-1\}$ would be prohibitively expensive. However, the diagonal block structure of the rotation matrices allows the matrix vector product to be rewritten as 2 element wise multiplication and 1 element wise addition instead

$$R^D_{\theta,k}x_k = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ \vdots \\ x_{D-1} \\ x_D \end{pmatrix} \otimes \begin{pmatrix} \cos k\theta_1 \\ \cos k\theta_1 \\ \cos k\theta_2 \\ \cos k\theta_2 \\ \vdots \\ \cos k\theta_{D/2} \\ \cos k\theta_{D/2} \end{pmatrix} + \begin{pmatrix} -x_2 \\ x_1 \\ -x_4 \\ x_3 \\ \vdots \\ -x_D \\ x_{D-1} \end{pmatrix} \otimes \begin{pmatrix} \sin k\theta_1 \\ \sin k\theta_1 \\ \sin k\theta_2 \\ \sin k\theta_2 \\ \vdots \\ \sin k\theta_{D/2} \\ \sin k\theta_{D/2} \end{pmatrix} \tag{83}$$

which makes RoPE computationally feasible.

14.3 Experiment Configuration

14.3.1 Applying RoPE to an SSP Based Model

As described above RoPE is designed to decay the attention value between two tokens the further they are from each other in the sequences. This theoretical functionality does not translate to the SSP mechanism. However, from an intuitive standpoint rotating each token with a gradually increasing angle appears as a reasonable approach to embed meaningful location information into the sequence without ruining structure in the data. In spirit with the location of the RoPE module in the transformer architecture/block, a similar approach will be employed in the Mamba-architecture. That is, RoPE will be applied in all blocks 9 right before the data enters the SSP module. Another justification for the choice of location is that positional information is supposed to enhance modeling along the sequence and as such should be located right before any module which performs temporal mixing, i.e. before the SSPs.

The employed RoPE implementation is a forked and modified version of this repository. Said implementation uses the efficient formulation given in equation (83), moreover the implementation minimized computation by caching the two vectors of sinusoidal and cosinusoidal in equation (83) as they are fixed.

14.3.2 Introducing Hyperparameters to RoPE

Assuming SSPs based models can be improved with positional information, they may however have a different requirement in regard to positional information than the fixed vanilla RoPE formulation proposes. To thoroughly investigate the possibility for any potential performance gain many aspects of RoPE are treated as tunable parameters and a wide range of experiments are conducted.

Note that RoPE as introduced and intended in the RoPE paper is entirely fixed with no hyperparameters. Nonetheless there are numerous ways to easily treat some of the RoPE components as tunable parameters. Whether the popular transformer models Llama, PaLM etc uses the vanilla variant of RoPE or some modified version is not immediately apparent and was not further investigated.

The following hyperparameters choices are empirically investigated:

- **The base**: The base of 10,000 used in equation (82) will be considered a binary parameter and may be either 10 or 10,000. Essentially the parameter controls how much each feature in some token embedding x_k is rotated as showcased in figure 12. It appeared odd that the default value of 10,000 rotates half of the features (from d=70-116 in the figure) of a token identically. On the contrary the base value of 10 ensures that all dimensions are rotated differently.
- **Trainable** θ: An additional binary parameter investigated is the choice to let θ be a learnable vector. The learnable variant is initialized in the same manner but is subsequently allowed to be updated through along the rest of the parameters in the model. Letting θ be learnable contributes a negligible amount of additional learnable parameters. For instance across the 6 blocks and with a token embedding dimension of 116 only amount to $2 \cdot 0.5 \cdot 116 \cdot 6 = 696$ additional parameters.
- **Normalization**: It is hypothesized that the rotation employed by RoPE is too intense for an SSP. Consequently, the rotation may be normalized along the sequence dimension. When applying RoPE the relative maximum rotation occurs between the first and the last token of sequence. The employed normalizing makes said maximum relative rotation equal an angle of 1 π radians. In other words, when normalized the first and last token will point in exact opposite direction.
- **Additional s6 options**: In the setting of the s6 SSP the parameters (B, C, Δ) materialize from parameterized functions of the input X. Hence more nuanced options arise in regard to exactly where RoPE can be applied. Maybe B (which dictates what enters the state) should be location aware, while C (which dictates what leaves the state) should not etc. Evidently any combination of B, C and Δ may either be a function of X or RoPE(X). Moreover the X which is processed by the s6 module may itself either be X or RoPE(X).

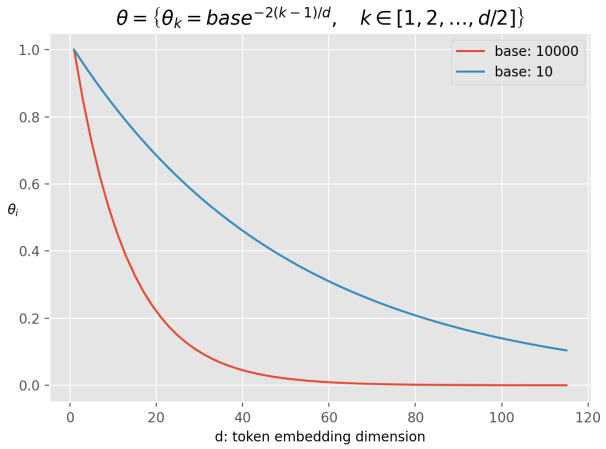

Figure 12: The effect of having 10 or 10,000 as the base in equation 82.

LRA Tasks Used. The effect of RoPE in all its hyperparameter configurations will be tested on two of the LRA tasks. Namely the IMDB task intended to represent an inherent categorical task and the Cifar10 in its numerical form to represent a task involving a numerical sequence. A Flattened image is not inherently particularly continuous along the sequence due to the flattening operation but nonetheless it is numerically valued. Thus it is by no means optimal but nonetheless the most appropriate of the LRA tasks. An audio or time series task would have been more ideal. The IMDB task will have its max sequence length reduced from 4096 tokens down to 1024 to reduce computation. In practice this only amounts to a small loss in quality as most reviews in the IMDB dataset are less than 1024 characters in the first place.

Models Employed. The LRA baseline results among other things showcased that the additional complexity of s4 did not entail a performance increase and thus s4 is deemed inferior to the much simpler s4d variant. Moreover the Mamba-block consistently outperformed the simple-block. As a result models based on the s4 SSP and models based on the simple-block are discarded from all further experiments. The models subject to the RoPE experiments are Mamba-s4d and Mamba-s6. Models trained on the IMDB task are trained for 15 epochs down from 35 and models trained on Cifar10-numerical are trained for 25 epochs down from 100. Except for the number of epochs, all model and optimizer hyperparameters remain identical to those used in the baseline LRA experiments (see table 2).

14.4 Results

Pilot experiments: On a side note before the main RoPE results are presented. The learnable positional embedding approach introduced in the original transformer paper was also attempted. In theory making the positional embeddings have no inductive bias and be entirely learnable should allow for maximal flexibility and customization in regard to the SSPs. This approach amounts to an increase in learnable parameters of $1024 \cdot 116 \cdot 6 = 712,704$, more than doubling the total parameter count. Performance wise the models would start to severely overfit and accuracy decreased 15-20 percent points. Subsequent experiments had the embedding weights tied across the 6 layers reducing the parameter count by a factor of 6 and solving the problem of overfitting. Nonetheless, the approach still negatively impacted performance by 5-10 percentage points and was discarded for further experimental work.

RoPE Configurations. Most but not all combinations of the variables present in listing 14.3.2 were used across the two LRA tasks. A total of 74 models were trained, 64 of which had some extent of RoPE positional embedding, while the remain 10 are considered the baseline. The below table displays the number of occurrence of each of the variables across the 2 tasks.

	s4d	Normed	Theta:10000	Trainable	B(s6)	C(s6)	Δ(s6)	X(s6)	NoPosEmb	Total
IMDB	13	18	11	13	13	10	10	8	5	37
Cifar10	16	13	17	13	12	11	11	8	5	37
Total	29	26	35	24	25	24	21	16	10	74
%	39%	35%	37%	32%	34%	28%	22%	27%	14%	

Table 5: The number of occurrences of each of the different rope hyper-parameters.

Each of the different RoPE configured models plotted against their corresponding base line models.

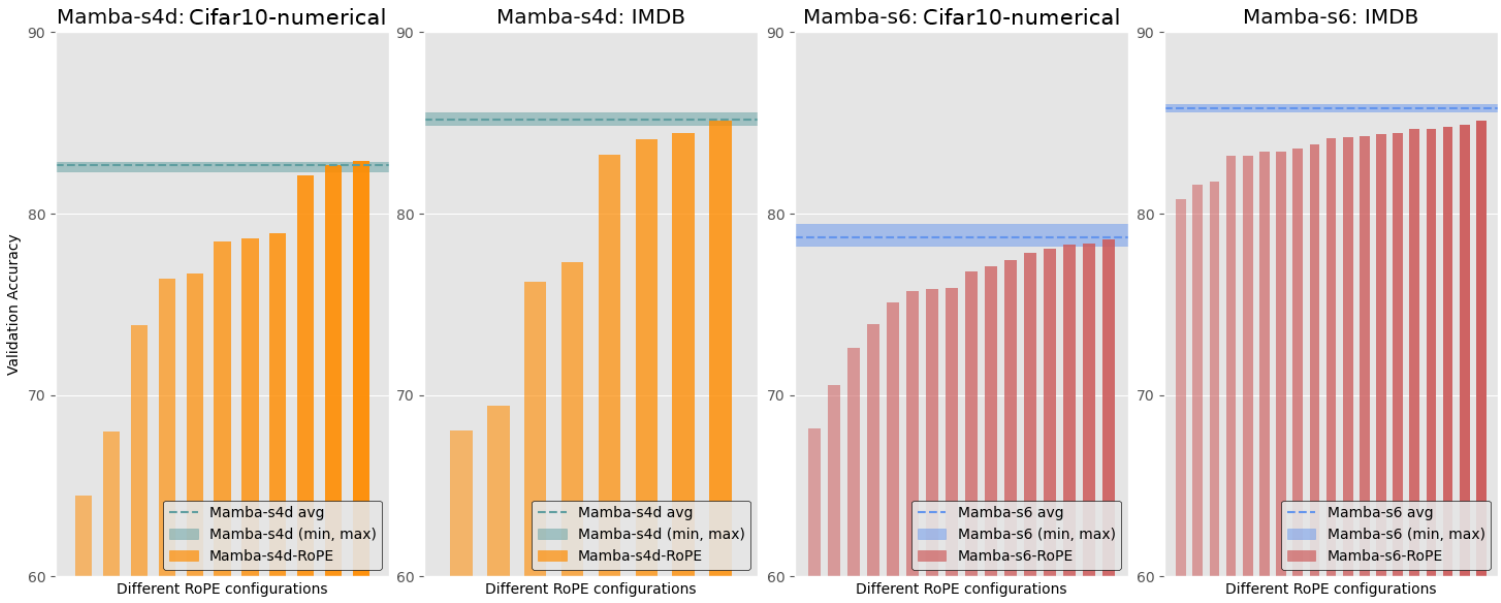

Figure 13: Each of the 4 plots corresponds to a different task/SSP combination. Each vertical bar in each plot is a 1 single run of a unique RoPE configuration (the width of the bar has no meaning). The 1 horizontal bar in each plot is the aggregated results from the 5 baseline models which do not use RoPE (the width is the min, max interval). In words, if a vertical bar was to cross the horizontal bar it would be an indication that RoPE improve performance. There are more since Mamba-s6 models since these have more configurations (B(s6), C(s6), $\Delta(s6)$, X(s6))

The 4 plots in figure 13 are the 4 different combinations of task/SSP. They are to be interpreted as follows: If any of the horizontal bars were to cross the vertical bar it would be evidence that "some" RoPE configuration performs better than the baseline models. See the figure caption for further details.

In the setting of s4d (the 2 left plots) a few RoPE configurations are able to reach the performance of the baseline and a single RoPE model manages to outperform the best baseline model with a few per-mille. In the instance of s6 (2 right plots), 3 RoPE configuration are able to reach the baseline models, but no RoPE model surpasses the mean-baseline (dashed horizontal line). In the s6 IMDB setting no RoPE model reaches the baseline models.

Across the 2 SSPs it appears that RoPE in general works better on the Cifar10 task than the IMDB task. Nonetheless, there does not seem to be substantial evidence that RoPE can improve the performance of an SSP based deep model. At best it appears RoPE simply does not harm the SSP.

14.4.1 Assessing the RoPE Parameters Individually

In an attempt to assess the impact of the different RoPE hyperparameters on an individual level, ordinarily least squares (OLS) regression is fitted on the 74 models. The explained variable being the accuracy and the explanatory variables being the RoPE hyperparameters. Additionally the dataset cifar/IMDB as well as SSP s6/s4d are included as control variables. The OLS regression includes an intercept/constant term (omitted in the plot) and all variables except accuracy are binary. To be explicit:

- IMDB=1 means the IMDB task while IMDB=0 is the Cifar10 task.
- s4d=1 means the s4d SSP is used while s4d=0 implies the use of s6 instead
- Normed=1 means the normalization procedure described in section 14.3.2 was employed and =0 means it was not.
- Theta:10000 =1 means the "base" was set to 10000 while Theta:10000 =0 means the base was set to 10. See figure 12 for clarification.
- Trainable=1 are instances where the RoPE parameters were allowed to be updated/learned beyond their initial values.
- $B(s6)$, $C(s6)$, Δ(x6) only relates to the s6 SSP. B(s6)=1 for instance corresponds to the setting where the input dependant B emerge from RoPE embedded sequence RoPE(X) while b(s6)=0 means the B emerge from the regular sequence X.
- similarly X(s6) only relates to s6 . X(s6)=1 implies that the sequence which enters the SSP for processing is RoPE embedded while X(s6)=0 is the regular non embedded sequence.
- It is emphasized that The B(s6), C(s6), Δ(x6) X(s6) variables are entirely independent and one does not imply/require one another.

The conducted OLS analysis possesses some reservations and any findings are at most indicative suggestions. One immediate shortcomings is the fact that the binary data contains dependencies. That is, if s4d=1 then all of (B(s6), C(s6), $\Delta(x6)$ X) are 0 and similarly if s4d=0 then at least one of (B(s6), C(s6), $\Delta(x6)$ X) are different from 0. Moreover while the OLS model contains an intercept it does not contain any interaction terms. While this could potentially have yielded more nuanced results it was omitted to avoid clutter caused by the blow up in total variables.

All OLS fitted coefficients along with their 95% confidence intervals are present in figure 14.

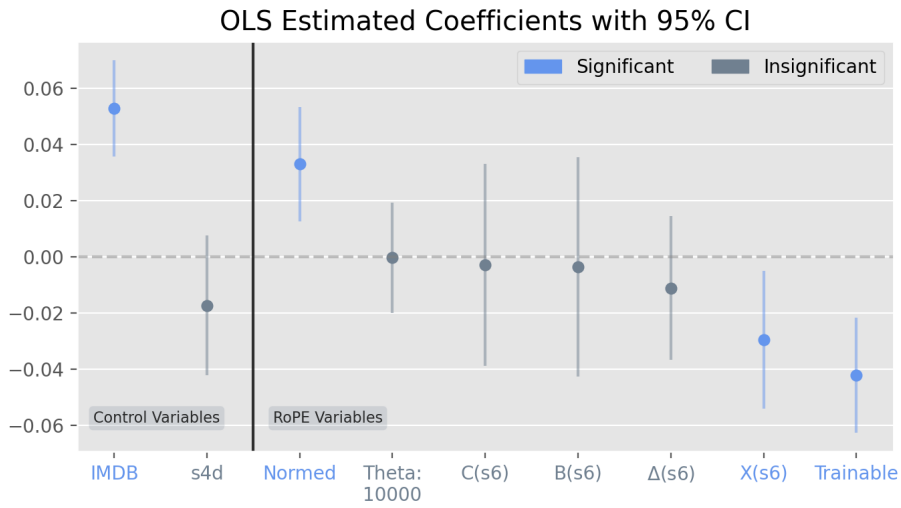

Figure 14: The OLS estimated coefficients for each of the binary variables. The intercept is omitted in the plot.

Consider first the OLS coefficients associated with the control variables in figure 14. The coefficient belonging to the IMDB variable indicates that on average the models score 5 percent-points higher on the IMDB task, which aligns with the findings from the previous section 13. Although not significant, the s4d coefficient suggests that across the 2 tasks s6 is the more performant SSP.

Consider now the OLS coefficients associated with the RoPE hyperparameters. Given the RoPE results presented previously (figure 13), it would have been safe to assume that all RoPE OLS coefficients would be around zero or negative. Nonetheless the "Normed" coefficient is positive and significant. One possible explanation has to do with the fact that the normalization is very aggressive, to the point where it eliminates the effect of RoPE. Recall that the normalization scales the rotation down such that the largest relative rotation (which occurs between the first and last token embedding) is exactly 1 π radians. That is, the normalization is potentially so severe that it nullifies the effect of RoPE entirely. Hereby the binary variable "Normed=1" corresponds to a setting where RoPE is present on paper but in practice is essentially disabled. In this sense "Normed=1" becomes a proxy variable for **no** positional embedding. This proposed hypothesis was not investigated experimentally. One way to do so, would be to: Fix all other variables and then gradually lessen the normalization even beyond the point where the normalization inverses and becomes an accentuating factor instead of a damping factor.

The remaining OLS RoPE coefficients follows the expected notion of being either zero or negative. For instance it appears that whether the "base" in the theta expression is 10 or 10000 is indifferent. One quantity which OLS suggests consistently deteriorates performance is letting the RoPE parameters be trainable. Letting RoPE be trainable should allow for greater flexibility but evidently cause a reduction in performance. One explanation is that the parameters are very sensitive and should have a reduced learning rate. Another explanation is that the learned values drift out outside the range of allowable θ values. This could be counteracted by some parameterization such that they are bounded and cannot escape the "meaningful" range of 0 to 1.

Lastly, consider the instance of the s6 specific hyperparameter. It appears that letting either (B, C or Δ) emerge as functions of RoPE embedded X matters little. On the contrary, letting the sequence processed by the SSP be RoPE embedded negatively impacts performance. It appears that RoPE causes greater harm when directly applied to the sequence, rather than indirectly through (B, C or Δ).

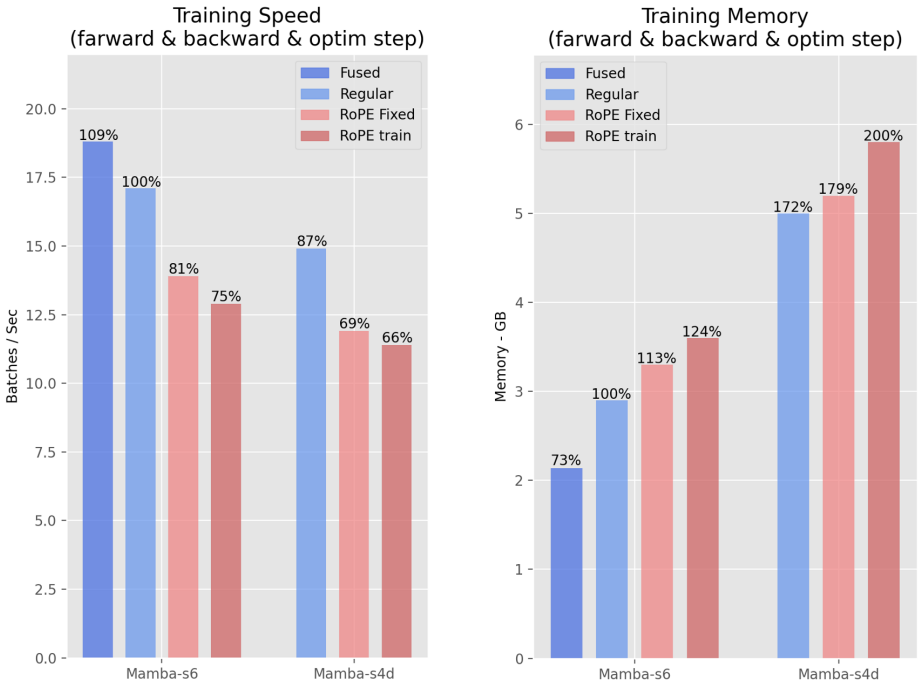

Figure 15: The %-quantities are relative to Mamba-s6-regular, which is considered the point of reference. The methodology used to produce the results are described in section 12.2.5.

14.4.2 Computational and Memory Overhead Implication of RoPE

Given the scale of the models employed in this thesis, RoPE only contributes an additional imperceptible 696 trainable or non-trainable parameters. Applying the methodology of only requiring the amount of parameters to be fixed would let the RoPE pass seemingly unnoticed on paper. Despite the small number of parameters, RoPE amounts to a large amount of additional computation. Consequently, it was deemed interesting to assess the computational and memory overhead induced by RoPE. The methodology to do so is described in section 12.2.5.

Consider figure 15 and disregard the fused-Mamba-s6 variant for now as it is merely a side result on its own and will be addressed momentarily. From the figure it is evident that RoPE across s4d and s6 substantially impacts training speed. Training memory is also impacted but to a lesser extent. Letting RoPE be learnable seems to imply a slight additional penalty to training speed while implying a more substantial additional penalty to the memory usage.

Training speed and training memory usage was deemed of primary interest. Nonetheless, inference speed and inference memory usage plots were also created and can be found in appendix C 17. During inference, RoPE entails an even greater speed reduction than at training, but causes no memory penalty at all. Whether RoPE is trainable obviously does not impact inference at all.

The overhead induced by RoPE is more severe than anticipated. One possible explanation has to do with the hardware efficiency of the implementation employed. The implementation uses the FLOP efficient RoPE formulation given in equation (83), however it is written in pure Pytorch. As a result, the performance overhead could be a result of Pytorch/python and may be substantially reduced by a custom low-level/cuda implementation.

Fused Mamba-s6 Notes. For completeness the fused Mamba-s6 version is included in figure 15. The details of the Mamba-s6 fused variant is described in section 12.1.1. Any model based on the Mamba block is applicable for a similar fused treatment, but would require the time and effort associated with the customized

low level implementation. Evidently, the fused Mamba s6 should only be compared to the regular Mamba-s6 (not RoPE s6) as it merely demonstrates a relative performance increase which could be implemented for any of the Mamba based models. Comparing the fused Mamba-s6 to the regular Mamba-s6 showcased that the fused variant is 9% faster and uses 27% less memory during training. Interestingly, the fused and non fused variant performs identical during inference as seen in the appendix C in figure 17.

14.5 Discussion

The immediate candidate for further experimental evaluation of positional information would be to investigate other concrete approaches beyond RoPE. Additionally, it could be of interest to investigate the implications of positional embedding in the setting of auto regressive generations and in general settings where the model is tested on longer sequences than it has seen during training. Especially the ability to generalize beyond the sequence length seen during training has been the downfall of the originally proposed absolute positional embedding approaches, and has been the motivation behind subsequent approaches such as Alibi[67].

Another approach to employ positional information, would be to only include it ones, rather than at each network block. The SSPs are not position agnostic as it is the case with self-attention. Additional positional information at a block level may induce too much unnecessary noise into the sequence, and should instead be treated as a one time undertaking, injected only after the token embedding layer.

Further investigation could also involve the inclusion of domain specific inductive bias. For instance, letting the positional embeddings reflect that an image has been flattened, by repeating the same embeddings for each row. However, flattened Cifar10 was never meant to be the optimal way to process images, but rather an attempt at requiring long range dependencies. The topic of processing images with sequential models is a whole field of research in itself involving vastly more sophisticated methods for incorporating spatial 2D information.

14.6 Conclusion

The results produced in this thesis in regard to positional embeddings suggests that the SSPs do not benefit from additional positional information. Although it can not be completely disproved considering the results are not significantly worse than the baseline, but merely equivalent or slightly worse. Additionally, only RoPE embeddings have been tested (although rather extensively), there may exist some other variation of positional embeddings which may improve performance, and the RoPE results produced here, do not necessarily generalize to other positional embedding approaches.

Nonetheless, the fact that positional embeddings appear to be ineffective in the setting of SSPs could be considered an argument in favor of SSPs: Not relying on positional embedding reduces the complexity of the network architectures and reduces the amount of computation required.

15 Investigating the Effect of Bidirectionality

This section experimentally investigates the potential increase in modeling performance by incorporating the models with bidirectionality. In comparison to positional embeddings, motivating the incorporation of bidirectionality is considerably more intuitive. The motivation is twofold, one related to causality and one related to fixed memory of recurrent models.

15.1 Motivation and Background

Bidirectionality remedies causality. All discussed SSPs so far have been causal. While causality is convenient for autoregressive generation it is an unnecessary constraint for classification tasks. Causality ensures that information only flows in one direction. Consequently, tokens later in the sequence may contain crucial information which explains or alters the interpretation of previous tokens, but causality prohibits the connection to be made. A caricatured example could be an IMDB review which states: "*this movie is so original and entertaining, if I was both blind and deaf*". In this instance the true ironic meaning is first revealed in second half of the sentence. However, causality ensures that the first half of the sentence may never be modeled according to true meaning. This becomes particularly problematic in the instance of sequence classification for the following reason: Recall that in the instance of a classification task, a `mean pooling` operation along the sequence is applied right before the decoder (see section 4.2.3). In the IMDB-review above, the true negative meaning may be averaged out by the positivity in the first half of the sentence. One potential solution is to exchange `mean pooling` in favour of only considering the last element in the sequence or a `CLS-token`. Even then the causal model would continue to model the first half of the IMDB review in an ignorant and suboptimal way. Moreover, only considering the final element for the classification may potentially mitigate the challenges induced by a causal model, but will only exacerbate issues related to the limited memory horizon discussed next. The obvious solution appears to let the model be bidirectional and keep the `mean pooling` operation approach.

Bidirectionality remedies the limited memory horizon. SSPs (and recurrent models in general) are subject to a fixed memory budget, due to their state representation. From a computation standpoint this is a strength but from a expressivity standpoint this may become a restriction. HiPPO theory briefly discussed in section 9 ensures efficient usage of the fixed memory budget by means of theoretically optimal compression of the sequence. Nonetheless compression implies loss of information and the greater the compression ratio the greater the loss. Furthermore the directional nature implies a notion of gradual memory loss also know as limited memory horizon[63] typically associated with the vanishing gradient problem: Recent tokens dominate the current state while past tokens gradually lose influence. This gradual memory loss is negatively biased towards earlier tokens in the sequence as they are gradually forgotten as the sequence is being processed. The notion of gradual memory loss is the reason final-token-classification is undesirable as the model prediction is biased towards the most recent tokens, that is the last tokens in the sequence. Bidirectionality does not solve the problem of fixed memory budget and gradual memory loss but it removes the notion of a first and last token of the sequence as both are considered both first and last. Consequently, "time" flows in both directions and the short term memory is not bias towards one end of the sequence, but equality bias or unbias towards both ends.

Bidirectionality aims to solve the problem of modeling non-causual relationships and alleviate bias towards tokens appearing later in the sequence.

Bidirectionality in the literature. The concept of bidirectional recurrent models such as bi-RNNs[73] and bi-LSTMs[42] are by no means a new invention. Nonetheless, bidirectionality as a concept plays a diminishingly small role in the recent body of literature which first introduced and then gradually improved the concept of SSPs. The body of literature under lead-author Albert Gu which proposed constitute HiPPO-theory, s4, s4d, s6[33][36][36][32]. Out of these the concept of bidirectionality is only ever briefly mentioned in the appendix of the s4d paper. Here it is stated that some tasks (LRA for instance) do not require a

causal model and as such bidirectionality should/is employed. No ablation studies are conducted to support the claim.

The above stated body of literature is more theoretical oriented and generally domain and modality invariant. Bidirectionality is more widely acknowledged and incoorporated in domain specific adaptations of SSP based modes. For instance in SSP based diffusion models[84], SSP based audio models[29] and SSP based DNA models[72]. The ablation studies related to isolating the effect of bidirectionality are entirely absent in the diffusion paper and kept to a minimum in the audio and DNA paper. In the latter two instances the number of trainable parameters are kept comparable, however no placebo model is employed to control for the shift in parameters (elaborated momentarily). Moreover, the increased memory and computation usage from the incorporation of bidirectionality is not assessed in either instance. While ablation studies assessing bidirectionality are not absent in the literature they are inadequate.

This thesis contributes to the assessment of bidirectionality in SSP based models in 3 ways:

- Through assessment of the performance on 2 LRA tasks. Moreover, 2 different SSPs are employed under comparable conditions, thus facilitating the assessment of bidirectionality on an SSP individual level.
- By controlling for the shift in parameters through the employment of a placebo model. By controlling for compute by employing a larger model which matches the wall-clock time of the bidirectional model.
- By examination of the implications with regard to memory and computation usage. Hereunder including a model where the computational budget is tied to that of a bidirectional.

15.2 Experiment Configuration

15.2.1 Equipping the Mamba Block with Bidirectionality

Inspired by the Mamba-s6-DNA[72] paper bidirectionality is incorporated into the Mamba block as follows.

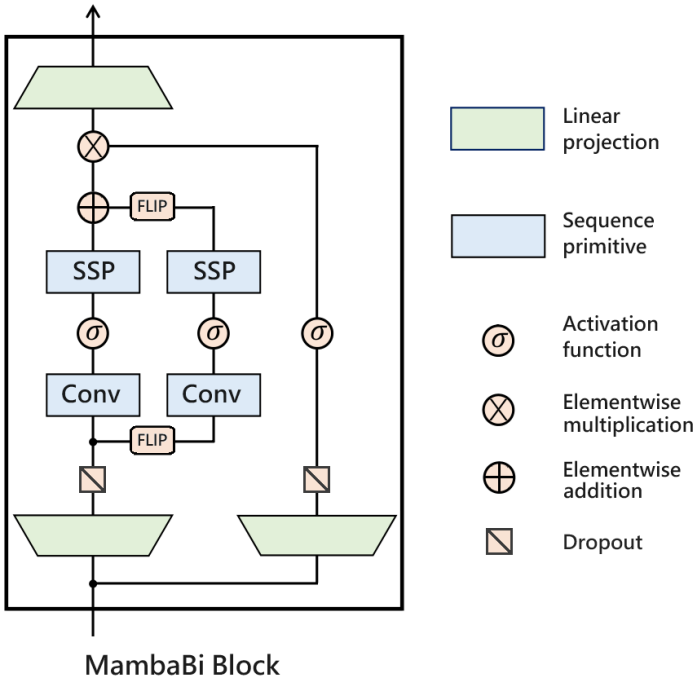

Figure 16: The MambaBi Block. The 2 SSP branches have independent parameters.

Consider the bidirectional Mamba block in figure 16: After the initial expansion projection the sequence is duplicated and split into 2 branches. The left branch processed as per usual by a causal 1d conv layer followed by an SSP layer. The right branch is flipped along the sequence dimension, then processed by a causal 1d-conv layer and a SSP layer and subsequently flipped along the sequence dimension again. The outputs of the right branch and left branch are then combined through element wise addition. The trainable parameters in the 1d-conv layer and the SSP layer of each branch are independent. Consequently the total number of trainable parameters in the model increases. To accommodate the increase in parameters the token embedding dimension of the bidirectional models is reduced accordingly such that the total number of parameters remain nearly identical. Recall that most of the parameters reside in the linear projection layers (see table 3) thus the embedding dimension only requires a slight reduction.

15.2.2 Including a Speed Matched Baseline

Even when the total amount of parameters are controlled for bidirectionality incur a substantial computational overhead. The exact details here off are elaborated and showcased in the result section. Considering the drastic overhead caused by bidirectionality it was deemed fit to include 2 additional models in the bidirectionality experiments. The 2 additional models are the result of increasing the size of the regular (non-fused) Mamba-s6 and regular Mamba-s4d until they match the training speed of their respective bidirectional counterparts. The 2 additional models serve to investigate whether the penalty to training speed (induced by bidirectionality) should have instead been utilized to simply make the models correspondingly larger. Model size is increased by increasing the token embedding dimension of the models. Consequently the "speed matched" Mamba-s6 receives an embedding dimension of 170 resulting in 1200k trainable parameters. The "speed matched" Mamba-s4d have less overhead and only receives an embedding dimension of 140 in turn yielding a total trainable parameter count of 820k.

15.2.3 Including a Placebo Baseline

The switch to a bidirectional model nearly doubles the amount of parameters residing in the sequence primitives of the model. To control for the fact that any potential performance improvement may be attributed to the shift in parameters rather than bidirectionality, a placebo model is introduced. The placebo model uses a placebo-mamba-block which is simply the bidirectional Mamba block with both "flipping" operations disabled. Consequently the placebo model has an identical distribution and number of parameters but remains directional and causal.

From a theoretical perspective the shift in parameters essentially corresponds to a shift of modeling emphasis from feature mixing to temporal mixing. If both the placebo model and the bidirectional performs better than the regular model, this may simply imply that the task requires greater emphasis for temporal mixing,

15.2.4 All Employed Models

Similar to the experimental section on positional embeddings only the s6 and s4d SSPs are employed. Each of the 2 SSPs appear in 4 different variants: regular, bidirectional, bidirectional-placebo and speed matched (SM).

Parameter count and token embedding dimension of the different Models Used in the bidirectional experiments can be seen below. The placebo models have identical token embedding dimension and parameter count as the s4d Bi and s6 Bi respectively.

	s4d	s4d Bi	s4d SM	s6	s6 Bi	s6 SM
Token Embedding Dimension	116	108	140	116	107	170
Total Trainable Parameters	575k	576k	816k	598k	597k	1200k

Table 6: Embedding dimension and total number of parameters for the different bidirectional models employed. The SM abbreviation stands for "speed matched". The total number of parameters stated above include the token embedding layer and the classification projection layer. As a result they vary slightly depending on the task. Concretely the above numbers originate from the IMDB review task.

15.2.5 Tasks and Hyperparameters

Identical to the experimental section on positional embeddings, the tasks used to facilitate the bidirectional experiments will be the IMDB task of length 1024 and the Cifar10-numerical task. Similarly all model and task hyperparameters remain identical, with minor reservation to token embedding dimension according to the above table. One slight deviation is the fact that for the bidirectional experiments all models are trained 5 times each, and their results are averaged and 95% confidence intervals are included.

15.3 Results

15.3.1 Compute and Memory Overhead of Bidirectionally

First the computation and memory overhead of bidirectionality is assessed, using the methodology described in section 12.2.5. As in the section on positional embeddings, the fused s6 variant should be disregarded in the comparison but has been included for completeness. The fused s6 variant is described in section 12.1.1 and a discussion and comparison of s6-fused versus regular s6 can be found in 14.4.2.

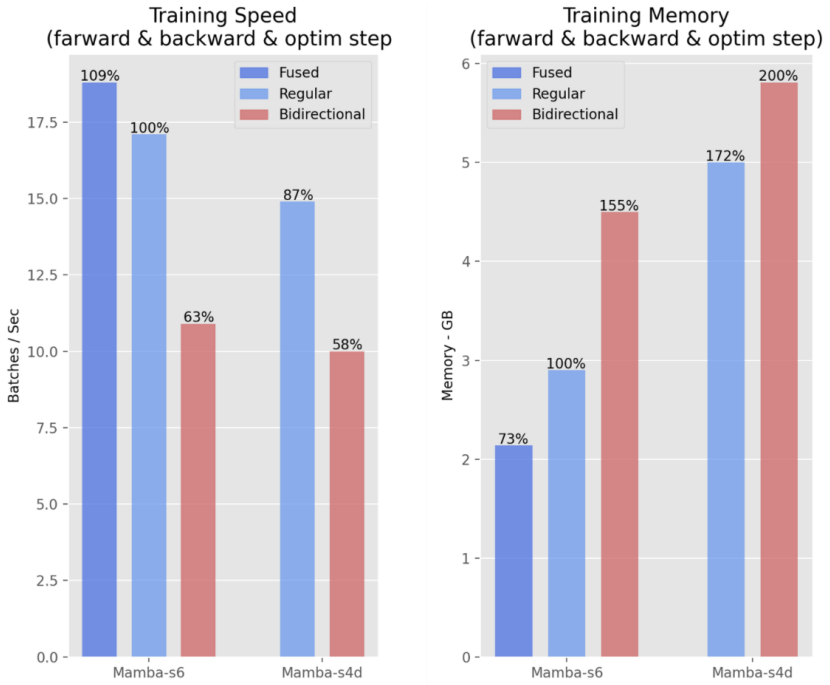

Figure 17: The %-quantities are relative to Mamba-s6-regular, which is considered the point of reference.

Figure 17 showcases that bidirectionality drastically reduces training speed and increases training memory consumption. s4d appears to be slightly more impeded than s6 with regard to relative reduction in training speed. s6 sees a substantial relative increase in memory consumption, where as s4d only uses slightly more memory when bidirectional. Corresponding plots in regard to inference memory and inference speed can be found in appendix C 17. The inference plots are practically equivalent to those of the training plots: the relative difference between the different models are the same the magnitude of the y-axis has just changed.

The results produced remain subject to reservations similar to those discussed in section 12.2.5. That is, the results are purely in terms of wallclock time and depend on the specific implementation and hardware. It was not investigated how computational overhead scales with sequence length or model size. Moreover no considerations or calculations were made in regard to the theoretical FLOP increase implied by bidirectionality.

15.3.2 Main Bidirectionality Results

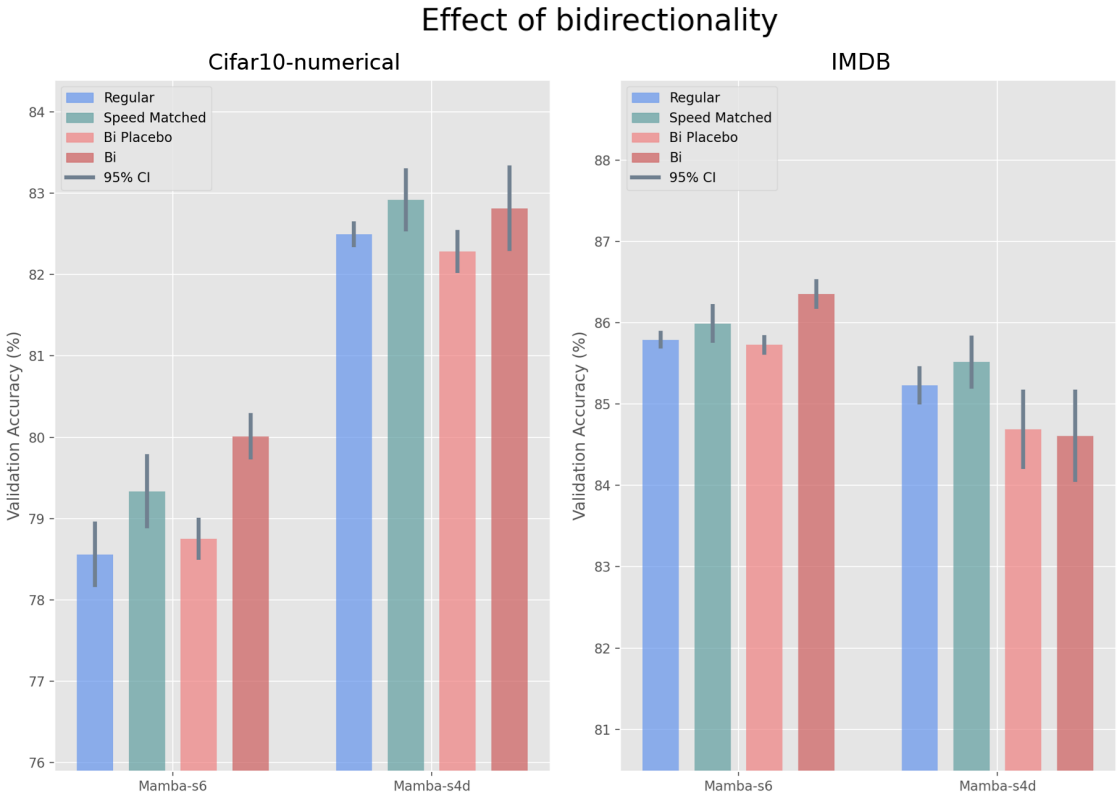

Figure 18: The effect of bidirectionality on the Cifar10-numerical and IMDB task. The bars are averages across 5 training runs and 95% CI are included.

Trends across models and tasks. The statistical significance is questionable due to the small sample size of only retraining all models 5 times. Nonetheless, one consistent trend is that the placebo models perform similar to the regular models (with one small reservation explained in the discussion section). Evidently, bidirectionality performance gains are not simply a consequence of shifting more parameters to the sequence primitives of the models. Furthermore the larger "speed matched" models consistently improves performance slightly. Interestingly in the setting of Mamba-s6, even doubling the amount of parameters only improves accuracy by less than 0.5%.

s4d: In the instance of Cifar10-numerical, bidirectionality appears to improve performance but only to a point of tying the performance of the "speed matched" variant. In the setting of the IMDB task both the placebo and the true bidirectional model performs worse than the regular model. Speculations as to why this occurred can be found in the discussion section below.

s6: On both tasks the bidirectional variant outperforms the regular version but also the "speed matched" variant. Based on the non overlapping confidence intervals, the bidirectional variant is significantly better than the regular version. Moreover it can *not be dismissed* that it too is better than the "speed matched" variant, however the proper statistical analysis was not conducted. The small sample size makes thorough statistical analysis unwarranted. The results strongly suggests that bidirectionality improves the modeling capabilities of the s6 SSP.

15.3.3 Limitations and Suggestions for Further Experimentation

Some limitations of the experiments conducted and the results produced are as follows. The two tasks are small toy problems and how the findings extrapolate to larger models and tasks remain unclear. Other approaches or extensions to incorporate bidirectionality into the Mamba-block were not investigated. For instance:

- Additional residual connections and/or normalization layers.
- A more sophisticated mixing mechanism than simply adding the output of the forward or reverse SSPs directly on top of each other.
- Additional independent/individual projection layers prior or post to each of the forward and reverse SSPs. One could imagine this would allow for greater flexibility and adaptability of each of the SSPs.
- Splitting the sequence along the feature dimension: Letting the forward SSP process the first $[0...d/2]$-features of the token embeddings while the reverse SSP processes the remaining $[d/2...(d-1)]$features. Subsequently the features are concatenated back onto of each other rather than added.
- In the instance of s4d, the FFT convolution could be computed without pre-padding the sequence and kernel such that the circular convolution overlaps. This would make the computation non causal and cut the the number of operations in half thus the making computation faster as well. Whether this would make the computation "bidirectional" per say is unclear.

It was not investigated how bidirectionality scales with sequence length. Potentially the sequences are simply too short for issues related to "limited memory horizon" to take affect. It may be the case that the size of the state is relatively large enough to simply memorize the whole sequence without the need to compress/discard information.

15.4 Discussion

Why did the bidirectional s4d perform worse than the regular s4d on the IMDB review task? The placebo model was introduced to control for the shift in parameters which is theoretically a shift in emphasis towards temporal mixing. Since both models perform worse than the regular model this highlights that for the IMDB task s4d actually requires less temporal mixing and more feature mixing. The reduction in performance may therefore be seen not as a consequence of bidirectionality but as a consequence of the reduction in feature mixing.

Feature mixing appears to be more crucial in the categorical setting of IMDB than the numerical setting of cifar. The reason as to why is unclear. In order to fully investigate this phenomenon an experiment would have to be conducted where; the amount of overall parameters are tied as well as the ratio between temporal mixing and feature mixing parameters are tied. This would require a different Mamba-Bi block for instance the one suggest above where: The sequence is split along the feature dimension and then letting the forward SSP process the first $[0...(d/2)]$ features of the token embeddings while the reverse SSP processes the remaining $[d/2...(d-1)]$ features. This bi-block was not implemented thus the experiment was not conducted.

Why did s6 seem to gain relatively more from bidirectionality? In line with the above discussion the immediate explanation as to why s6 gains more from bidirectionality than s4d, may simply come down to: s6 does not gain more from bidirectionality s6 simply loses less from the reduction in feature mixing. One reason could be that while the s6 is SISO and primarily performs temporal mixing, the sequence transformation performed by s6 does also entail some amount of feature mixing as highlighted in section 7.5.2. Thus when more parameters are shifted into s6, the overall feature mixing (of the deep model) is less impacted than when s4d undergoes a similar parameter shift.

Do SSPs not forget? Finally it is noted that, due to its selection mechanism, s6 should in theory be the superior to s4d in regard to efficient usage of the hidden state memory. The selection mechanism was

originally motivated as a means to improve state-memory efficiency by only "selecting" relevant information and thus not letting noise pollute the state. Evidently if the limited memory horizon is in fact a problem s6 should be less impacted due to its theoretical superior state-memory-efficiency. In other words bidirectionality (theorized to mitigate the limited memory horizon) should have a greater positive impact on s4d than s6. This is not the case. This implies that SSPs do not suffer from the limited memory horizon in the first place. If this is indeed the case it poses a strong argument for SSP based models general capabilities. But the results produced in this thesis are far from conclusive in this regard and only mildly suggestive. One way to investigate whether they do not forgot would be to remove the `mean pooling` operation and solely base the decoding on the final token embedding of the sequence. If accuracy does not degrade from this modification it implies that the final token embedding is able to summarize/represent the whole sequence.

15.5 Conclusion

Bidirectionality induces substantial overhead in regard to both computation speed and memory consumption. It should not be taken for granted just because the task allows for a bidirectional model, as it appears to be the case in the literature. Consequently, bidirectionality should be used with caution especially if you are on a limited computation budget: In particular if the use of bidirectionality implies a reduction in feature mixing. However in a setting where increasing model size does not further improve performance and computational budget is secondary, bidirectionality appears a potential source of further performance. Finally the results mildly suggest that the SSPs do not suffer from limited memory horizon and that bidirectionality only serves to alleviate the causality of the SSPs.

16 Exploring Approaches to Bidirectional Pretraining

This section seeks to experimentally investigate the effect of pretraining SSPs based models before applying them to downstream tasks. In particular, it is of interest to investigate *alternative pretraining approaches* when the downstream task allow for bidirectional models.

16.1 Pretraining Methodologies

The pretraining approach of choice in the literature when using non causal models is masked language modeling (MLM), also sometimes referred to as BERT-style pretraining. On the contrary, the preferred retraining choice when employing a causal model is next token prediction (NTP) which is sometimes referred to as GPT-style pretraining. A thorough introduction and discussion of the 2 approaches is found in section 4.3.

16.1.1 Pretraining Proposals

Upon their introduction it was hypothesized that NTP is a more potent pretraining approach. The argument being that MLM produces a weaker loss/error signal. Recall that MLM masks (typically) 15% of the tokens for prediction. On the contrary, NTP allows the model to predict all but one token in the sequence. In this sense, on a per sequence/observation basis, NTP allows for almost 7 times the amount of predictions, which intuitively would produce a more meaningful error signal. This, in turn, should imply more accurate gradients and in turn a more meaningful model update per training step.

This notion raises the question: Given a non-causal task, should you use a causal model to allow for NTP pretraining and then subsequently morph the pretrained model into a bidirectional one for downstream finetuning? To investigate this question 2 alternative pretraining approaches are proposed:

- **F1**: Perform NTP pretraining. Subsequently, at the finetuning stage, duplicate the learned SSP of each block and place the copy in a separate branch along with adequate flipping operations such that the Mamba-bi block from figure 16 emerges. This approach is dubbed 1F, to indicate that one forward model was used during pretraining.
- **1F1R**: The other approach is dubbed 1F1R. This name is meant to indicate that 2 separate models are NTP pretrained. One forward model similar to the first approach. Additionally, 1 reverse (1R) model is pretraining. The reverse model is exclusively trained on reversed sequences. Before the finetuning stage the 2 models are fused into one as follows: Let all linear projection layers as well as the encoding and decoding layer be those of the forward pretrained model. Extract the conv1d layer as well as the SSP of the reverse pretrained model and use those for the reverse branch in the Mamba-bi blocks. Evidently the reverse pretrained model has only ever seen data in reverse order and should have proper inductive bias when injected into the reverse branch.

16.2 Adopting the Species Classification Task

To test the proposed F1 and F1R1 pretraining approaches a new dataset is adopted as testbed. The dataset originates from a 2023 DNA-sequence modeling paper by **[Nguyen et al. 2023]**. The model proposed in the paper is termed HyenaDNA and excels at modeling long range dependencies in DNA and was at the time of publication the state of the art model. The sequence primitive at the core of HyenaDNA responsible for its effective long range capabilities is a data dependent global convolution. That is, the parameters of the global convolution kernel depend on the specific input sequence, making it somewhat comparable to s6. In contrast to s6 but similar to s4 and s4d, the global convolution is evaluated using the convolution theorem. In this sense HyenaDNA may be viewed as a hybrid between s4/s4d and s6. The use of the convolution theorem implies log-linear scaling in sequence length and allows them to model DNA sequences up to a length of 1 million tokens.

At the time of publication (2023) HyenaDNA demonstrated state of the art performance in a variety of benchmarks but has since been surpassed by both regular Mamba-s6 and a specialized MambaDNA-s6[72].

The benchmark of particular interest is a 5 way species classification. That is, given a string of DNA, predict the belonging class among the options: human, lemur, pig, mouse and hippo. What makes this problem interesting is an ablation study with regard to both sequence length and pretraining. Here the HyenaDNA paper showcase that the longer the DNA sequence the better HyenaDNA performs. More importantly as they train on longer sequences they observe that pretraining becomes progressively more crucial for optimal performance. Their concrete results in this regard can be seen in table 7.

Evidently, the 5 way species classification task appears a promising testbed to examine the effect of different pretraining approaches given the fact that it previously has been shown to exhibit strong scaling with pretraining. Moreover the dataset is flexible and allows the length of the training sequences to easily be adjusted to an arbitrary length.

The findings from the previous section revealed that the inclusion of bidirectionality only led to a minuscule performance gain on the LRA tasks. As a result bidirectionality became difficult to justify given the additional memory and compute overhead. However it speculated that the LRA sequences are too short for bidirectionality to become important, further warranting the need for a new testbed.

16.3 Experiment Configuration

The species dataset in detail. The dataset comprises all chromosomes from the 5 respective species and can be downloaded from this database. The experiments conducted in this thesis uses the exact dataloader and training splits employed in the official HyenaDNA code repository. The train, validation and test splits are on a per chromosome level, such that (subject to slight variations) chromosome 1, 4, 7 constitute the test set while 2, 5, 8 constitute the validation set and the remaining chromosomes are the training set. The data sampling process proceeds as follows: Given a split and a sequence length L, uniformly sample a species and subsequently a chromosome. Uniformly sample an index from the entire length of the chromosome. Extract a slice of length L starting from the index. If the slice exceeds the maximum length of the chromosome it is adequately padded with a special token.

Only the s6 SSP will be employed. The s6 paper[32] showcases excellent results on a slightly altered 5 way species classification task. While the foundation remains identical they choose to let all 5 species be from the great ape family. Consequently the results are not comparable. However the purpose of this section is pretraining. Pretraining on species classification is not ablated in the s6 paper, thus the HyenaDNA species task was picked in favour of the great ape version from the s6 paper. Nonetheless s6 has proven itself an excellent DNA classification model and s6 will be the only SSP employed for the pretraining experiments. The uni and bidirectional model sizes are are carried over from the previous section. The exact values are found in table 6.

Hyperparameters and pretraining specifications. Experiments are conducted on sequences of lengths (1024, 2048, 4096, 8192, 16384, 32768). Batch size is reduced from 64 down to 32 to avoid memory limitations when training on sequences of length 32768. A moderate dropout value of 0.1 is adapted for the species classification task. The value was not tuned.

- **Fine tuning specifics:** During finetuning learning rate is set to 1e-3 but scaled according 1024/L where L is the sequence length. The data sampling procedure described in section 16.3 renders the concept of epochs inadequate. The models are finetuned on 100,000 DNA samples i.e. for 3125 steps. After this amount of training steps validation accuracy has converged.
- **Pretraining specifics:** Pretraining lasts for twice the amount of steps as finetuning, that is 6250. Subsequently the model is finetuned using the procedure stated above. Pretraining uses more aggressive learning rates with a base learning rate of 4e-3 which is only scaled proportional to 2048/L. During pretraining weight decay and dropout is set to 0. In accordance with HyenaDNA, pretraining is only performed on the training set belonging to the Human species class.

- **MLM specifics:** In accordance with the specifications proposed in the original MLM paper[22]: 15% of the tokens are selected for prediction. Of the 15%, 80% are replaced with the special MASK-token, 10% are replaced with a random token from the vocabulary and the remaining 10% are left untouched.

16.4 Establishing Baseline Results on the Species Task

The proposed F1 and F1R1 pretraining methods relies on 2 crucial assumptions. 1: Bidirectionality improves model performance on the species task. 2: Pretraining improves model performance on the species task. If either of these assumptions are not met F1 and F1R1 become meaningless. Consequently, a baseline is established to verify the assumptions and relate the performance of Mamba-s6 to the results reported in the HyenaDNA paper. The baseline consists of 3 model types. A regular causal model, a NTP pretrained causal model and a non-pretrained bidirectional model. Training the 3 models on the 6 different sequence lengths is summarized by the content of figure 19. The final Mamba-s6 validation accuracies at sequence lengths comparable to what is used in the HyenaDNA paper are reported in table 7.

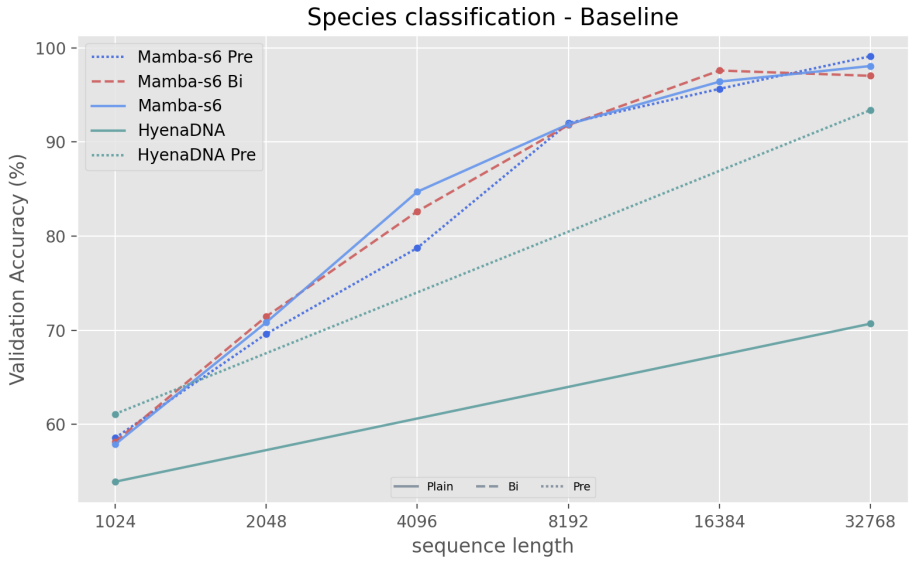

Figure 19: Species baseline results. The x axis is not to scale. Notice that the line style indicate pretraining and bidirectionality.

L	Mamba-s6	Mamba-s6-Bi	Mamba-s6-Pre	Mamba-s6 (avg)	HyenaDNA†	HyenaDNA-pre†
1k	57.9	58.1	58.5	58.2	53.9	61.1
33k	98.1	97.0	99.1	98.0	70.7	93.4
262k					71.4	97.9

Table 7: Final validation accuracies. The HyenaDNA results are not reproduced, they are recited from the paper. The HyenaDNA paper does not report results for sequence lengths (2048, 4096, 8192, 16384) thus they have been omitted from the table to reduce clutter. Mamba-s6 avg is simply the average across the 3 different s6 models. Due to reaching 98% accuracy at 33k it was deemed unnecessary to train Mamba-s6 on any longer sequences.

16.4.1 Accessing the 2 Assumptions

The results displayed by figure 19, are by no means definitive. However, when attributing fluctuations among the 3 Mamba models to the fact that they were only trained ones, all 3 models perform close to identical.

There is no trend among them, no model is consistently performing below or above the other. When facing Mamba-s6 with the species task, bidirectionality nor pretraining matters. In conclusion neither assumption is met. Section 16.5 addresses how the assumptions are eventually met.

16.4.2 Mamba-s6 Compared to HyenaDNA

Preliminary side note regarding the comparison. HyenaDNA at 1k and 33k uses a smaller model (in parameter count) than Mamba-s6 while Hyena at 256k uses a substantially larger model. Across all sequence lengths HyenaDNA is trained for significantly more steps than the Mamba-s6.

From figure 19 it is evident that the objective of classifying species from DNA strings requires sufficiently long strings of DNA in order to be adequately solvable. Mamba-s6 displays an excellent ability to make use of the gradually increasing sequence lengths which in turn implies its capabilities as a DNA model. From table 7 at sequences length of 33k it Mamba-s6 has reached near perfect accuracy and essentially "solved" the species task. At this sequence length Mamba-s6 substantially beats HyenaDNA. Moreover, at 33k Mamba-s6 matches the performance of HyenaDNA trained on sequences of length 262k. Interestingly HyenaDNA is slightly better than Mamba-s6 at the 1k length. This may be attributed to the longer training time of HyenaDNA or the smaller HyenaDNA model actually being an advantage at this length. Nonetheless, the 1k mark is uninteresting considering the species task fundamentally requires longer sequence lengths to be sufficiently solved.

16.5 Making Pretraining Matter

In the experimental section on bidirectionality, it was found that bidirectionality had a vanishingly small but still positive impact on performance. Evidently, it would not come as a surprise if bidirectionality had no impact on certain tasks such as the current species classification. However, the concept of pretraining is widely acknowledged and is essential to the success of many aspects of modern deep learning. Moreover, pretraining was found to be crucial for this very task by the HyenaDNA paper. In the investigation as to why pretraining had no impact in the Mamba-s6 setting the following areas were examined through pilot experiments:

- The models simply did not pretrain long enough.
- The models should pretrain on longer sequences than what they were subsequently finetuned on.
- Only the output projection layer should be trained during finetuning.
- Finetuning should use a drastically reduced learning rate across all layers in order to not ruin/forgot the patterns learned during pretraining.
- The models should be allowed to pretrain on data from all 5 species and not only the human genome.

None of the above focus areas appeared to have any impact. Consequently, none were adopted except for the last proposal due to the simple fact that it appeared odd to not pretrain on data from all species.

16.5.1 Making the Species Task More Difficult

Eventually it was examined if the species classification task was simply too easy. This notion was motivated by the fact that at sequences length of 33k all Mamba-s6 models achieve close to perfect validation accuracy. Furthermore, the original Mamba-s6 paper adapted a more difficult variation of the species classification task. This could likely be a result of the authors deemed the HyenaDNA version too easy.

At this point, it was deemed too late to make the transition to the great ape variant used in the s6 paper. However, one final option remained, which was to make the current species dataset more difficult. The straightforward approach to do so was to reduce the amount of fine tuning data. The gradual reduction in finetuning data proceed as follows: The training set of all species were split into 2 subsets. One exclusively for finetuning and one exclusively for pretraining. (Up until this point pretraining had only been conduced on the training set corresponding to the human genome.)

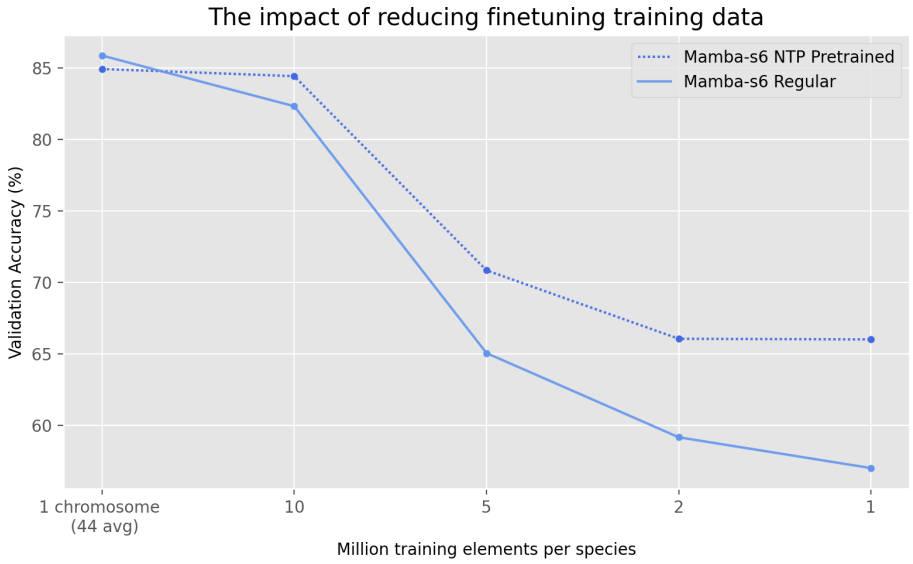

Figure 20: The training runs use a sequence length of 8k and are conducted according to the training recipe covered in 16.3. The number 44 million is the length of 1 chromosome per species, averaged across the 5 species.

- The first attempt at reducing the finetuning data would have 3 chromosomes present in the finetuning set while the remaining were used for pretraining.
- Subsequently finetuning was limited to the smallest chromosome per species, but Mamba-s6 remained unaffected.
- Not until the finetuning set was reduced to the first 10,000,000 elements of a single chromosome per species did s6 become considerably impacted. Performance as function of finetuning data can be seen in figure 20.

Figure 20 strongly suggest that as finetuning data become gradually more scarce pretraining becomes correspondingly more important. Evidently the scarce data setting makes the problem more difficult which in turn facilitates pretraining to become impactful. Thus the pretraining assumption has been meet and the main topic of this section may proceed. i.e. how the proposed F1 and F1R1 approaches compare to the regular NTP and MLM approaches. Some reservations remain in regard to the 2. assumption as bidirectionally has not (yet) been experimentally confirmed to improve performance on the species classification task.

16.6 Main Results

The experiments conducted to investigate the effect of pretraining, bidirectionality and the proposed F1 and F1R1 methods uses a finetuning set of 1 chromosome per species. Of that one chromosome per species only the first 1,000,000 elements are used. This is identical to the right most observations in 20. All remaining chromosomes (not including test or validation chromosomes) are used for pretraining. The experiments are conducted using the 3 sequence lengths (4096, 8192, 16384). Notice that slicing 1,000,000 elements consecutively using a slice length of 16.384 produces 61 unique slices. In other words, there are in principle only 61 unique samples per class when using a length of 16k. The pretraining and finetuning procedure remains what is described in section 16.3.

16.6.1 Results: Limited Finetuning Setting

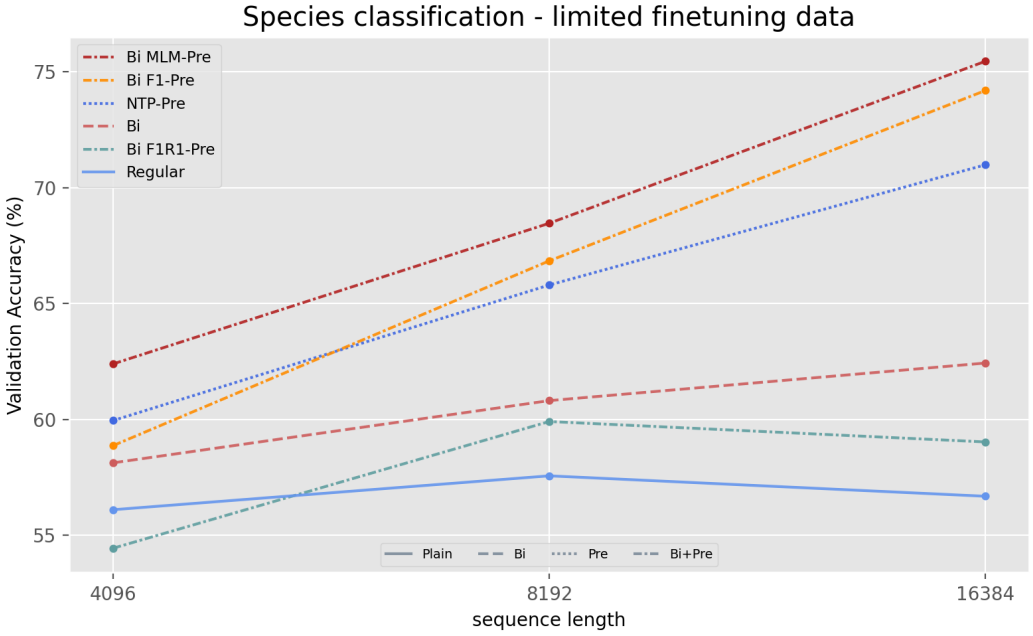

Figure 21: All models are Mamba-s6 models in a variety of configurations implied by the names in the legend. To help navigate the plot, notice the legend is sorted according to best average accuracy (averaged across the 3 sequence lengths). Additionally notice that the line style indicates bidirectionality and pretraining.

Due to the limited finetuning data all models reach a training accuracy of 95+%. Nonetheless validation accuracy does not overfit: during finetuning validation accuracy swiftly stagnates but does not reach a point of degradation.

Consider the different models in figure 21. In the following observational analysis F1R1 (green line) is disregarded as it is considered and outlier. It will instead be addresses separately afterwards. The following trends across models can be observed:

- Previously the sole factor which determined model performance was the sequence length on which the models were finetuned. Given the now severely limited data setting the regular Mamba-s6 (blue dense line) has not only severely degraded in performance it appears to degrade as sequence lengths increase.
- Bidirectionality consistently improves performance across the 3 sequence lengths and re-enables to models to scale with sequence lengths.
- To an even greater degree than bidirectionality, pretraining appears to improve performance across all sequence lengths and it too re-enables models to scale with sequence lengths.
- The performance gain and scaling behavior induced by bidirectionality and pretraining appears to be additive: The 2 best models are the once which possess both features.

16.7 Discussion

16.7.1 Why is Bidirectionality Suddenly Important?

The baseline DNA experiments with the plentiful finetuning dataset seen in figure 19 suggests bidirectionality is not important. On the contrary the very scarce finetuning dataset setting seen in figure 21 has bidirectionally (dashed red line) consistently perform better than the baseline model (dense blue line) across

the 3 sequence lengths. How and why bidirectionally suddenly becomes important in the data scarce setting is unclear and was not investigated. It could be that bidirectionally introduces some amount of regularizing which lets the model generalize better.

16.7.2 The Potential of F1

With some reservations in regard to the fact that the different models were only trained ones and the results are **not** the average from multiple runs: The best performing model is the conventional MLM pretrained bidirectional model (19% point over the Mamba-s6-regular at 16k length). The runner up is the proposed F1 (18% point over the Mamba-s6-regular at 16k length) verifying the potential of the suggested approach. One advocate for the F1 approach is in regard to its substantial computational advantage during pretraining. As indicated by figure 17 from the previous section, bidirectionality significantly impacts training speed. However the F1 model is pretrained as a causal model and thus avoids the computational overhead induced by bidirectionality. Moreover during pretraining F1 has reduced parameter count: Only by the stage of finetuning when its SSPs are duplicated does F1 reach the full parameter count prescribed in table 6. As a result F1 pretrains in half the wall-clock-time of the native bidirectional model, while almost matching it in performance on the downstream task. An interesting future experiment would be to allowed the F1 model to pretrain for the same amount of wall clock time as the MLM pretrained model in order to see if it would close the gab.

The success of F1 implies another potential use case. Consider a situation where a large directional model has undergone substantial (and expensive) pretraining. At some point the practitioner may want to employ the model on a downstream task which allows for bidirectionality. In this particular instance, the pretrained directional model may be morphed into a bidirectional one. The results produced in this thesis suggest that this would improve performance while avoiding the need to redo pretraining.

16.7.3 F1R1, why did it not work?

This question was not investigated experimentally. Nonetheless, one likely intuitive explanation is as follows. Recall the SISO formalization utilized by all SSPs described in this work. Given the F1 part of the F1R1, consider now the state space (in some network block) responsible for modeling the 1st feature of the token embeddings of the whole sequence. After pretraining the learned time scale parameter Δ_k[5] associated with this spate space may emphasize short range dependencies. Subsequently the R1 model is pretrained, however in this instance the learned timescale parameter associated with the 1st feature emphasizes long range dependencies. When the SSPs of the 2 models are morphed into one bidirectional model a disagreement in regard to the timescale of the first dimension arises and the forward and reverse model end up counteracting each other. This notion obviously extends to all features in the token embedding and not just the 1st dimension. Similar counteracting behavior does **not** occur in the pure F1 model. In this instance after pretraining, the forward and reverse SSPs are identical and completely agree on exactly what timescale each feature has, avoiding the issue entirely.

16.7.4 Proposals to make F1R1 work

One potential solution to accommodate the F1R1 time scale disagreement could be: Perform the usual NTP pretraining on F1. Subsequently, rather than pretraining the R1 from scratch, instead initialized its weights to be that of the pretrained F1 model. Ideally all dimensions have settled for a suitable timescale after the F1 pretraining, in which they remain for the duration of the R1 pretraining.

Another approach which may alleviate disagreement and increase adaptability between the forward and reverse part of the F1R1 would be to extend each of them with an individualized linear projection: Prior to the conv1d layer in the Mamba-block another linear projection and nonlinear activation is inserted. When the model is morphed bidirectional, the forward and reverse SSPs retain their respective individual projection layer. Hopefully the additional projection layer allows for additional adaptability and flexibility. Obviously this approach substantially increases the total parameter count. The token embedding dimension would have to be decreased further if the total parameter count is to remain fixed.

[5]It is noted that although Δ is input dependant but the analogy should still hold

Neither of the above described proposals were tested and are left as potential further work.

16.8 Conclusion

For sequence lengths of 33k Mamba-s6 reaches an accuracy of 98% and essentially "solves" the DNA classification task. Moreover Mamba-s6 substantially beats the SOTA baseline model HyenaDNA. Mamba-s6 does not appear to require pretraining nor bidirectionality, in order to performant on the task. The DNA task is made artificially difficult by substantially limiting the available finetuning data. In this limited setting both bidirectionally as well as pretraining becomes important. The performance of the F1R1-pretraining method produces underwhelming results. However the proposed F1-pretraining method produces the 2. best performing model while only requiring half the wall-clock time to pretrain, highlighting the potential of F1.

17 Conclusion

Theoretical. The state space primitive (SSP) was theoretically derived in full. Starting from a perspective of simulating the discrete sequence according to a latent continuous-time, higher-order, ordinary differential equation (ODE). The ODE was first decomposed into a system of first order equations, and subsequently imposed to follow the classical state space model (SSM). By further assuming time invariance, a well behaved continuous-time, linear-time-invariant (LTI) system emerged. From there, exact zero order hold discretization of the continuous-time LTI SSM was derived in full, by which the discrete recurrent formulation of the SSP emerged. The discrete recurrence could be unrolled to form a global kernel, which allows for the entire sequence to be simulated by a single global convolution computation. Thus, the SSP emerges in a dual recurrent/convolutional formulation, which utilizes the hardware friendly global convolution during training, while employing the recurrent formulation during auto regressive inference.

Subsequently, the SSP was thoroughly interpreted and discussed, in particular with respect to its most distinguishable characteristic, its learnable discretization parameter. This parameter was showcased as a modulating quantity, which allows for the inherent timescale of the data to be learned, and could also be thought of as a parameter which allows the length of the convolutional filters to be learned. Furthermore, it was displayed how a simple gated recurrence could emerge from discretizing a 1D state space, which implies that the discretization procedure may be viewed as a learned gating mechanism. Finally, concerns regarding the linearity of the SSP in regard to the expressivity of the overall deep model were addressed: By means of a simple comparison to the CNNs and by highlighting the results from a recent paper, which theoretically shows that simple linear RNNs can be arbitrarily expressive.

Computational. The theoretically derived SSP is not directly applicable in a deep model. Thus, 3 different approaches to make it computationally viable were showcased, each of which amounted to a different but concrete/practical SSP (s4, s4d and s6) ready to be used in a deep model. It was highlighted that a common denominator among the 3 SSPs is to utilize a single input single output (SISO) formulation, which employs d different state space. That is, one for each of the features in the token embedding vector. To achieve further computational feasibility, all SSPs impose structure on the state transition matrix A. s4d assumes a diagonal A, which trivializes the global kernel construction. On the contrary, s4 employs a DPLR, which requires an overly complicated but still efficient construction of the global kernel. The parameters of s6 are input dependent, thus time variant, which disqualifies the usage of the global convolution. As a result, it was demonstrated in full how s6 instead relies on an associative parallel scan instead. Finally, it is exemplified that the data dependent nature of s6 entails some extent of feature mixing to occur. This is in contrast to the setting of s4 and s4d, where the d employed state spaces operate entirely independently of each other and exclusively perform temporal mixing.

Empirical. The experimental section first sought to establish that key accuracy metrics on the widely adopted long range arena (LRA) benchmark could be met. With minor reservation to the LRA itself, all 3 SSPs aligned with the expectation set from their respective papers. Moreover, it was revealed that the Mamba network architecture was particularly performant, which also suggested that emphasis should be on feature mixing rather than temporal mixing when employing deep state space models.

Motivated by their importance in the context of self-attention, it was thoroughly investigated whether positional embeddings could improve the prediction accuracy of s4d and s6. The popular rotational positional embeddings (RoPE) were adopted and implemented in a flexible manner, which allowed for them to be highly configurable. A total of 64 RoPE configurations across 2 different deep state space models and 2 different LRA tasks were tested. The results produced hereby did not indicate that RoPE led to an improvement of the deep state space models.

Motivated by the inherent directionality of the deep state space models, it was sought to investigate if making them bidirectional could lead to improved prediction accuracy. Bidirectionality makes the model non-causal but *should* also improve their memory horizon, which is how far back in the sequence they can remember. The testing methodology was particularly meticulous and ensured to control for: 1) number of total trainable

parameters, 2) the number of parameters present inside the SSP versus elsewhere in the deep model, 3) the computational overhead introduced from making the model bidirectional. The results produced from the experiments indicate that bidirectionality (slightly) improves performance of the deep state space models. Furthermore, the results suggest that bidirectionality merely solves the problem of causality and the models do not struggle with a *limited memory horizon* in the first place.

Finally, *alternative methods* to pretraining bidirectional state space models were investigated. Motivated by the theoretically superior next token prediction (NTP) pretraining approach, this thesis proposed to pretrain a directional model using NTP, then subsequently, (right before downstream finetuning), morph the model bidirectional. The proposed pretraining approach empirically (almost) matches the conventional bidirectional pretraining method. However, the proposed approach only used half the wall-clock time to pretrain compared to the conventional. Furthermore, the proposed approach showcases that, already pretrained directional models, may in general, be converted bidirectional without the need to redo pretraining.

References

[1] State-space representation - wikipedia. URL https://en.wikipedia.org/wiki/State-space_representation.

[2]

[3] Cauchy matrix - wikipedia. URL https://en.wikipedia.org/wiki/Cauchy_matrix.

[4] Controller discretization, .

[5] Discretization - wikipedia, . URL https://en.wikipedia.org/wiki/Discretization.

[6] Euler method - wikipedia. URL https://en.wikipedia.org/wiki/Euler_method.

[7] Kalman filter - wikipedia. URL https://en.wikipedia.org/wiki/Kalman_filter.

[8] Root of unity - wikipedia. URL https://en.wikipedia.org/wiki/Root_of_unity.

[9] Vandermonde matrix - wikipedia. URL https://en.wikipedia.org/wiki/Vandermonde_matrix.

[10] Woodbury matrix identity - wikipedia. URL https://en.wikipedia.org/wiki/Woodbury_matrix_identity.

[11] M. Beck, K. Pöppel, M. Spanring, A. Auer, O. Prudnikova, M. Kopp, G. Klambauer, J. Brandstetter, and S. Hochreiter. xlstm: Extended long short-term memory. 5 2024. URL http://arxiv.org/abs/2405.04517.

[12] I. Beltagy, M. E. Peters, and A. Cohan. Longformer: The long-document transformer. 4 2020. URL http://arxiv.org/abs/2004.05150.

[13] G. E. Blelloch. Prefix sums and their applications, 1990.

[14] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei. Language models are few-shot learners. 5 2020. URL http://arxiv.org/abs/2005.14165.

[15] T.-C. Chi, T.-H. Fan, P. J. Ramadge, and A. I. Rudnicky. Kerple: Kernelized relative positional embedding for length extrapolation. 5 2022. URL http://arxiv.org/abs/2205.09921.

[16] T.-C. Chi, T.-H. Fan, A. I. Rudnicky, and P. J. Ramadge. Dissecting transformer length extrapolation via the lens of receptive field analysis. 12 2022. URL http://arxiv.org/abs/2212.10356.

[17] T.-C. Chi, T.-H. Fan, and A. I. Rudnicky. Attention alignment and flexible positional embeddings improve transformer length extrapolation. 11 2023. URL http://arxiv.org/abs/2311.00684.

[18] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. 12 2014. URL http://arxiv.org/abs/1412.3555.

[19] T. Dao. Flashattention-2: Faster attention with better parallelism and work partitioning. 7 2023. URL http://arxiv.org/abs/2307.08691.

[20] S. De, S. L. Smith, A. Fernando, A. Botev, G. Cristian-Muraru, A. Gu, R. Haroun, L. Berrada, Y. Chen, S. Srinivasan, G. Desjardins, A. Doucet, D. Budden, Y. W. Teh, R. Pascanu, N. D. Freitas, and C. Gulcehre. Griffin: Mixing gated linear recurrences with local attention for efficient language models. 2 2024. URL http://arxiv.org/abs/2402.19427.

[21] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. *ImageNet: A Large-Scale Hierarchical Image Database*. IEEE, 2009. ISBN 9781424439911.

[22] J. Devlin, M.-W. Chang, K. Lee, K. T. Google, and A. I. Language. Bert: Pre-training of deep bidirectional transformers for language understanding, 2018. URL https://github.com/tensorflow/tensor2tensor.

[23] S. K. Dhall and S. Lakshmivarahan. Parallel computing using the prefix problem. 1994.

[24] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. 10 2020. URL http://arxiv.org/abs/2010.11929.

[25] J. L. Elman. Finding structure in time. *Cognitive Science*, 14:179–211, 3 1990. ISSN 0364-0213. doi: 10.1207/s15516709cog1402_1.

[26] D. Y. Fu, H. Kumbong, E. Nguyen, and C. Ré. Flashfftconv: Efficient convolutions for long sequences with tensor cores. 11 2023. URL http://arxiv.org/abs/2311.05908.

[27] F. Gloeckle, B. Y. Idrissi, B. Rozière, D. Lopez-Paz, and G. Synnaeve. Better faster large language models via multi-token prediction. 4 2024. URL http://arxiv.org/abs/2404.19737.

[28] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. URL http://www.iro.umontreal.

[29] K. Goel, A. Gu, C. Donahue, and C. Ré. It's raw! audio generation with state-space models. 2 2022. URL http://arxiv.org/abs/2202.09729.

[30] Google. Palm 2 technical report, 2023.

[31] A. Gu. Modeling sequences with structured state spaces a dissertation submitted to the department of department of computer science and the committee on graduate studies of stanford university in partial fulfillment of the requirements for the degree of doctor of philosophy, 2023. URL https://purl.stanford.edu/mb976vf9362.

[32] A. Gu and T. Dao. Mamba: Linear-time sequence modeling with selective state spaces, 2023. URL https://github.com/state-spaces/mamba.

[33] A. Gu, T. Dao, S. Ermon, A. Rudra, and C. Ré. Hippo: Recurrent memory with optimal polynomial projections, 2020. URL https://github.com/HazyResearch/hippo-code.

[34] A. Gu, K. Goel, and C. Ré. Efficiently modeling long sequences with structured state spaces. 10 2021. URL http://arxiv.org/abs/2111.00396.

[35] A. Gu, I. Johnson, K. Goel, K. Saab, T. Dao, A. Rudra, and C. Ré. Combining recurrent, convolutional, and continuous-time models with linear state-space layers. 10 2021. URL http://arxiv.org/abs/2110.13985.

[36] A. Gu, A. Gupta, K. Goel, and C. Ré. On the parameterization and initialization of diagonal state space models. 6 2022. URL http://arxiv.org/abs/2206.11893.

[37] Y. Gu, O. Obeya, and J. Shun. Parallel in-place algorithms: Theory and practice. 3 2021. URL http://arxiv.org/abs/2103.01216.

[38] A. Gupta, A. Gu, and J. Berant. Diagonal state spaces are as effective as structured state spaces. 3 2022. URL http://arxiv.org/abs/2203.14343.

[39] S. Haque, Z. Eberhart, A. Bansal, and C. McMillan. A survey of deep learning and foundation models for time series forecasting. volume 2022-March. IEEE Computer Society, 2024. ISBN 9781450392983. doi: 10.1145/nnnnnnn.nnnnnnn.

[40] R. Hasani, M. Lechner, T.-H. Wang, M. Chahine, A. Amini, and D. Rus. Liquid structural state-space models. 9 2022. URL http://arxiv.org/abs/2209.12951.

[41] S. Hochreiter and J. Schmidhuber. Lstm. 2007.

[42] Z. Huang, W. Xu, and K. Yu. Bidirectional lstm-crf models for sequence tagging. 8 2015. URL http://arxiv.org/abs/1508.01991.

[43] A. Ivanov, N. Dryden, T. Ben-Nun, S. Li, and T. Hoefler. Data movement is all you need: A case study on optimizing transformers. 6 2020. URL http://arxiv.org/abs/2007.00072.

[44] A. Q. Jiang, A. Sablayrolles, A. Roux, A. Mensch, B. Savary, C. Bamford, D. S. Chaplot, D. de las Casas, E. B. Hanna, F. Bressand, G. Lengyel, G. Bour, G. Lample, L. R. Lavaud, L. Saulnier, M.-A. Lachaux, P. Stock, S. Subramanian, S. Yang, S. Antoniak, T. L. Scao, T. Gervet, T. Lavril, T. Wang, T. Lacroix, and W. E. Sayed. Mixtral of experts. 1 2024. URL http://arxiv.org/abs/2401.04088.

[45] A. Katharopoulos, A. Vyas, N. Pappas, and F. Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention. 6 2020. URL http://arxiv.org/abs/2006.16236.

[46] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. URL http://code.google.com/p/cuda-convnet/.

[47] R. E. Ladner and M. J. Fischer. Parallel prefix computation, 1980.

[48] S. Li, C. You, G. Guruganesh, J. Ainslie, S. Ontanon, M. Zaheer, S. Sanghai, Y. Yang, S. Kumar, and S. Bhojanapalli. Functional interpolation for relative positions improves long context transformers. 10 2023. URL http://arxiv.org/abs/2310.04418.

[49] Y. Li, T. Cai, Y. Zhang, D. Chen, and D. Dey. What makes convolutional models great on long sequence modeling? 10 2022. URL http://arxiv.org/abs/2210.09298.

[50] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. Dollár. Microsoft coco: Common objects in context. 5 2014. URL http://arxiv.org/abs/1405.0312.

[51] Y.-C. LIN and L.-L. HUNG. *New Parallel Prefix Algorithms*. WSEAS, 2009. ISBN 9789604741076.

[52] Y. Liu, Y. Tian, Y. Zhao, H. Yu, L. Xie, Y. Wang, Q. Ye, and Y. Liu. Vmamba: Visual state space model. 1 2024. URL http://arxiv.org/abs/2401.10166.

[53] Z. Liu, H. Mao, C.-Y. Wu, C. Feichtenhofer, T. Darrell, and S. Xie. A convnet for the 2020s. 1 2022. URL http://arxiv.org/abs/2201.03545.

[54] X. Ma, C. Zhou, X. Kong, J. He, L. Gui, G. Neubig, J. May, and L. Zettlemoyer. Mega: Moving average equipped gated attention. 9 2022. URL http://arxiv.org/abs/2209.10655.

[55] E. Martin and C. Cundy. Parallelizing linear recurrent neural nets over sequence length. 9 2017. URL http://arxiv.org/abs/1709.04057.

[56] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. 1 2013. URL http://arxiv.org/abs/1301.3781.

[57] T. Munkhdalai, M. Faruqui, and S. Gopal. Leave no context behind: Efficient infinite context transformers with infini-attention. 4 2024. URL http://arxiv.org/abs/2404.07143.

[58] E. Nguyen, M. Poli, M. Faizi, A. Thomas, C. Birch-Sykes, M. Wornow, A. Patel, C. Rabideau, S. Massaroli, Y. Bengio, S. Ermon, S. A. Baccus, and C. Ré. Hyenadna: Long-range genomic sequence modeling at single nucleotide resolution. 6 2023. URL http://arxiv.org/abs/2306.15794.

[59] A. Oppenheim and G. Verghese. Properties of lti state-space models 5.1 introduction, .

[60] A. Oppenheim and G. Verghese. Properties of lti state-space models 5.1 introduction, .

[61] A. Orvieto, S. De, C. Gulcehre, R. Pascanu, and S. L. Smith. Universality of linear recurrences followed by non-linear projections: Finite-width guarantees and benefits of complex eigenvalues. 7 2023. URL http://arxiv.org/abs/2307.11888.

[62] A. Orvieto, S. L. Smith, A. Gu, A. Fernando, C. Gulcehre, R. Pascanu, and S. De. Resurrecting recurrent neural networks for long sequences. 3 2023. URL http://arxiv.org/abs/2303.06349.

[63] R. Pascanu, T. Mikolov, and Y. Bengio. On the difficulty of training recurrent neural networks. 11 2012. URL http://arxiv.org/abs/1211.5063.

[64] B. Peng, E. Alcaide, Q. Anthony, A. Albalak, S. Arcadinho, S. Biderman, H. Cao, X. Cheng, M. Chung, M. Grella, K. K. GV, X. He, H. Hou, J. Lin, P. Kazienko, J. Kocon, J. Kong, B. Koptyra, H. Lau, K. S. I. Mantri, F. Mom, A. Saito, G. Song, X. Tang, B. Wang, J. S. Wind, S. Wozniak, R. Zhang, Z. Zhang, Q. Zhao, P. Zhou, Q. Zhou, J. Zhu, and R.-J. Zhu. Rwkv: Reinventing rnns for the transformer era. 5 2023. URL http://arxiv.org/abs/2305.13048.

[65] B. Peng, D. Goldstein, Q. Anthony, A. Albalak, E. Alcaide, S. Biderman, E. Cheah, X. Du, T. Ferdinan, H. Hou, P. Kazienko, K. K. GV, J. Kocoń, B. Koptyra, S. Krishna, R. McClelland, N. Muennighoff, F. Obeid, A. Saito, G. Song, H. Tu, S. Woźniak, R. Zhang, B. Zhao, Q. Zhao, P. Zhou, J. Zhu, and R.-J. Zhu. Eagle and finch: Rwkv with matrix-valued states and dynamic recurrence. 4 2024. URL http://arxiv.org/abs/2404.05892.

[66] M. Poli, S. Massaroli, E. Nguyen, D. Y. Fu, T. Dao, S. Baccus, Y. Bengio, S. Ermon, and C. Ré. Hyena hierarchy: Towards larger convolutional language models. 2 2023. URL http://arxiv.org/abs/2302.10866.

[67] O. Press, N. A. Smith, and M. Lewis. Train short, test long: Attention with linear biases enables input length extrapolation. 8 2021. URL http://arxiv.org/abs/2108.12409.

[68] Z. Qin, X. Han, W. Sun, B. He, D. Li, D. Li, Y. Dai, L. Kong, and Y. Zhong. Toeplitz neural network for sequence modeling. 5 2023. URL http://arxiv.org/abs/2305.04749.

[69] Z. Qin, S. Yang, and Y. Zhong. Hierarchically gated recurrent neural network for sequence modeling. 11 2023. URL http://arxiv.org/abs/2311.04823.

[70] Z. Qin, S. Yang, W. Sun, X. Shen, D. Li, W. Sun, and Y. Zhong. Hgrn2: Gated linear rnns with state expansion. 4 2024. URL http://arxiv.org/abs/2404.07904.

[71] T. K. Rusch and S. Mishra. Unicornn: A recurrent model for learning very long time dependencies. 3 2021. URL http://arxiv.org/abs/2103.05487.

[72] Y. Schiff, C.-H. Kao, A. Gokaslan, T. Dao, A. Gu, and V. Kuleshov. Caduceus: Bi-directional equivariant long-range dna sequence modeling. 3 2024. URL http://arxiv.org/abs/2403.03234.

[73] M. Schuster and K. K. Paliwal. Bidirectional recurrent neural networks, 1997.

[74] I. W. Selesnick and C. S. Burrus. Fast convolution and filtering, 1999.

[75] J. T. H. Smith, A. Warrington, and S. W. Linderman. Simplified state space layers for sequence modeling. 8 2022. URL http://arxiv.org/abs/2208.04933.

[76] J. Su, Y. Lu, S. Pan, A. Murtadha, B. Wen, and Y. Liu. Roformer: Enhanced transformer with rotary position embedding. 4 2021. URL http://arxiv.org/abs/2104.09864.

[77] Y. Sun, L. Dong, S. Huang, S. Ma, Y. Xia, J. Xue, J. Wang, and F. Wei. Retentive network: A successor to transformer for large language models. 7 2023. URL http://arxiv.org/abs/2307.08621.

[78] Y. Tay, M. Dehghani, S. Abnar, Y. Shen, D. Bahri, P. Pham, J. Rao, L. Yang, S. Ruder, and D. Metzler. Long range arena: A benchmark for efficient transformers. 11 2020. URL http://arxiv.org/abs/2011.04006.

[79] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, D. Bikel, L. Blecher, C. C. Ferrer, M. Chen, G. Cucurull, D. Esiobu, J. Fernandes, J. Fu, W. Fu, B. Fuller, C. Gao, V. Goswami, N. Goyal, A. Hartshorn, S. Hosseini, R. Hou, H. Inan, M. Kardas, V. Kerkez, M. Khabsa, I. Kloumann, A. Korenev, P. S. Koura, M.-A. Lachaux, T. Lavril,

J. Lee, D. Liskovich, Y. Lu, Y. Mao, X. Martinet, T. Mihaylov, P. Mishra, I. Molybog, Y. Nie, A. Poulton, J. Reizenstein, R. Rungta, K. Saladi, A. Schelten, R. Silva, E. M. Smith, R. Subramanian, X. E. Tan, B. Tang, R. Taylor, A. Williams, J. X. Kuan, P. Xu, Z. Yan, I. Zarov, Y. Zhang, A. Fan, M. Kambadur, S. Narang, A. Rodriguez, R. Stojnic, S. Edunov, and T. Scialom. Llama 2: Open foundation and fine-tuned chat models. 7 2023. URL http://arxiv.org/abs/2307.09288.

[80] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. 6 2017. URL http://arxiv.org/abs/1706.03762.

[81] A. R. Voelker, I. K. Kaji´c, and C. Eliasmith. Legendre memory units: Continuous-time representation in recurrent neural networks, 2019. URL https://github.com/abr/neurips2019.

[82] J. Wang, T. Gangavarapu, J. N. Yan, and A. M. Rush. Mambabyte: Token-free selective state space model. 1 2024. URL http://arxiv.org/abs/2401.13660.

[83] R. L. Williams and D. A. Lawrence. *Linear state-space control systems*. John Wiley Sons, 2007. ISBN 9780471735557.

[84] J. N. Yan, J. Gu, and A. M. Rush. Diffusion models without attention. 11 2023. URL http://arxiv.org/abs/2311.18257.

[85] S. Yang, B. Wang, Y. Shen, R. Panda, and Y. Kim. Gated linear attention transformers with hardware-efficient training. 12 2023. URL http://arxiv.org/abs/2312.06635.

[86] Y. Yang, Z. Xing, C. Huang, and L. Zhu. Vivim: a video vision mamba for medical video object segmentation. 1 2024. URL http://arxiv.org/abs/2401.14168.

[87] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. Salakhutdinov, and Q. V. Le. Xlnet: Generalized autoregressive pretraining for language understanding. 6 2019. URL http://arxiv.org/abs/1906.08237.

[88] B. Zhang and R. Sennrich. Root mean square layer normalization. 10 2019. URL http://arxiv.org/abs/1910.07467.

[89] L. Zhu, B. Liao, Q. Zhang, X. Wang, W. Liu, and X. Wang. Vision mamba: Efficient visual representation learning with bidirectional state space model. 1 2024. URL http://arxiv.org/abs/2401.09417.

Appendix A

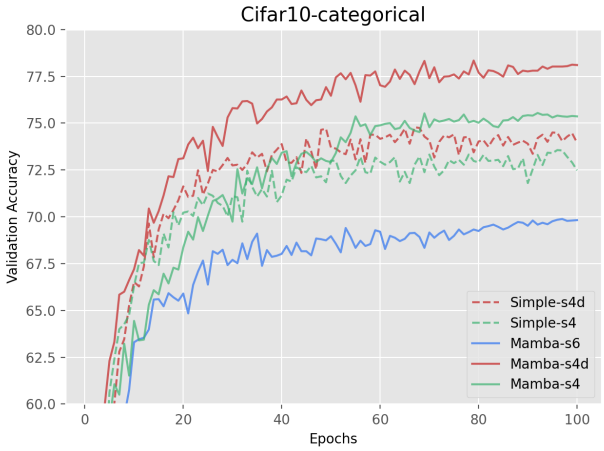

Figure 22: LRA Baseline Cifar10 categorical learning curves.

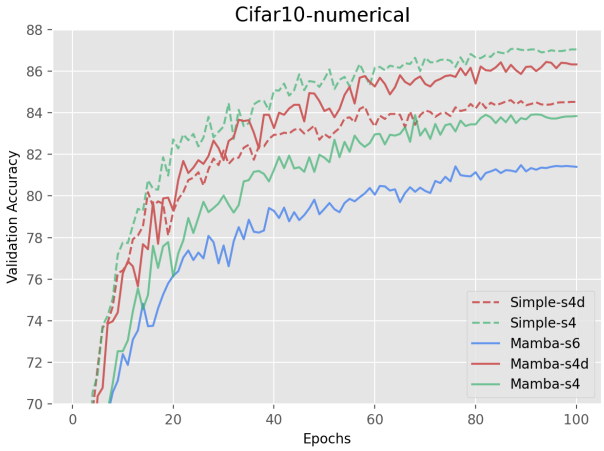

Figure 23: LRA Baseline Cifar10 continuous learning curves.

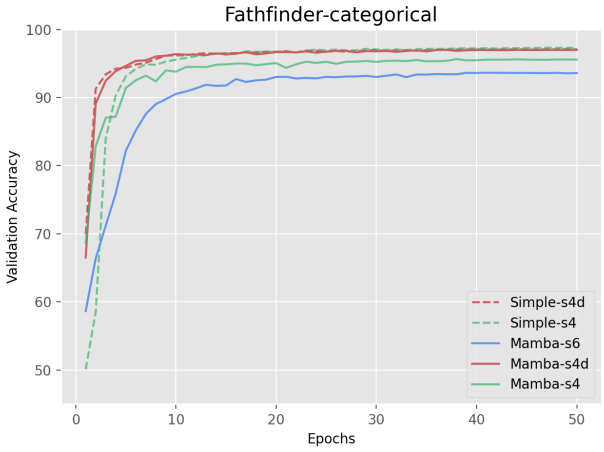

Figure 24: LRA Baseline Fathfinder categorical learning curves.

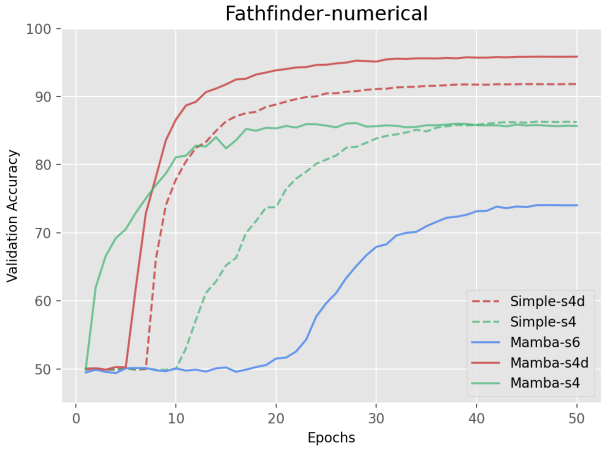

Figure 25: LRA Baseline Fathfinder continuous learning curves.

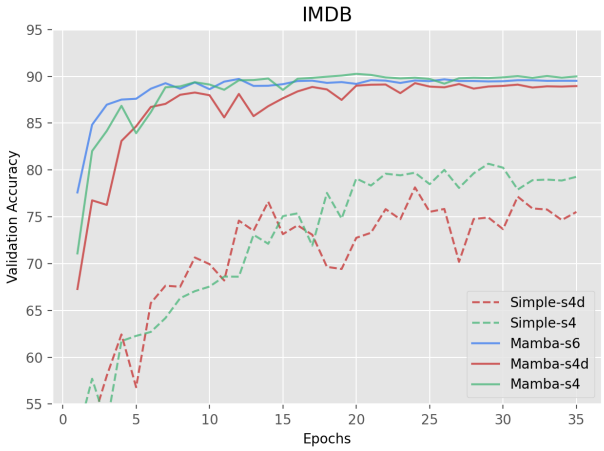

Figure 26: LRA Baseline IMDB learning curves.

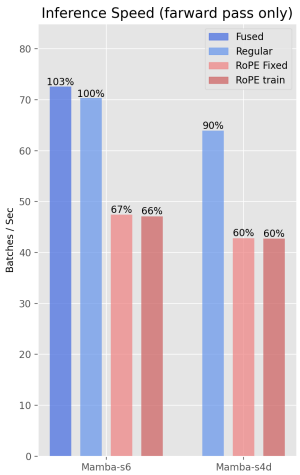

Figure 27: RoPE inference speed overhead comparison.

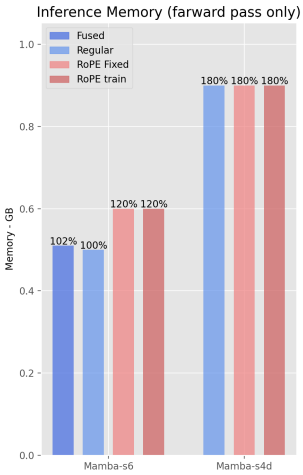

Figure 28: RoPE inference memory overhead comparison.

Inference speed and memory overhead of RoPE.

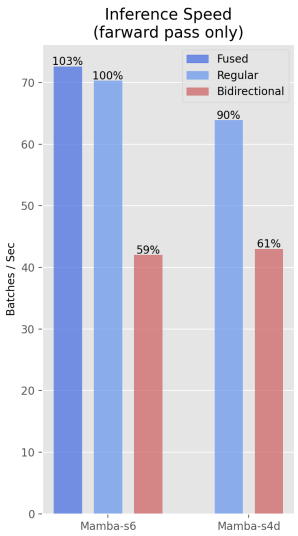

Figure 29: Bidirectional inference speed overhead comparison.

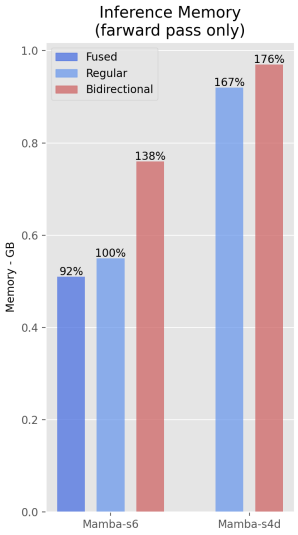

Figure 30: Bidirectional inference memory overhead comparison.

Inference speed and memory overhead of Bidirectionality.