How Does a Machine Learn Sequences: an Applied Mathematician's Guide to Transformers, State-Space Models, Mamba, and Beyond

Annan Yu
Center for Applied Mathematics, Cornell University

October 22, 2024

Outline of This Tutorial

1. (First Hour) Part I: A Survey of Sequential Models
2. (Second Hour) Part II: A Deep Dive into State-Space Models

Outline of Part I

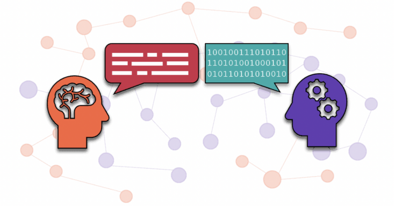
1. Introduction to sequential models
2. Recurrent units and related models
3. More advanced sequential models

Introduction to Sequential Models

Sequential Data in Real World

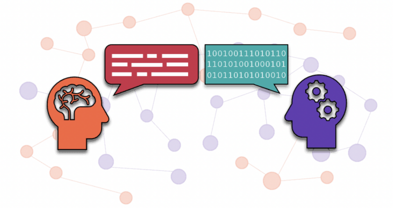
Sequential Data in Real World

Natural Language Processing

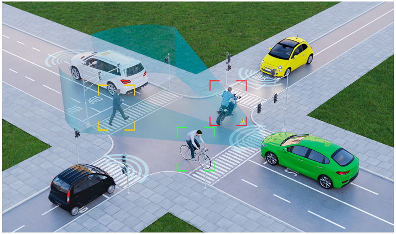


Sequential Data in Real World

Natural Language Processing

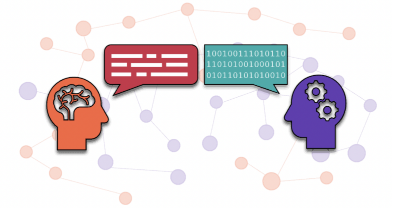


Computer Vision

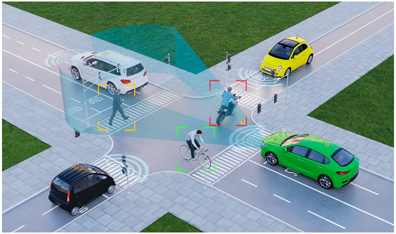


Sequential Data in Real World

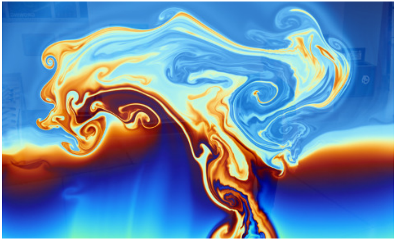
Natural Language Processing



Computer Vision

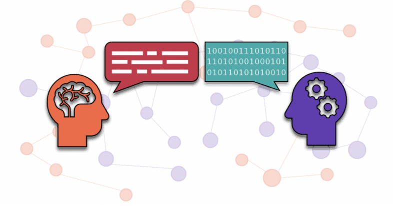


Scientific Applications

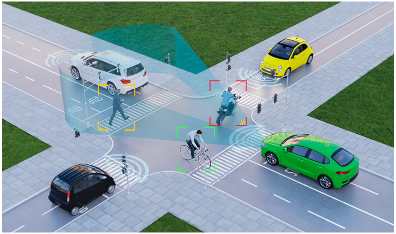


Sequential Data in Real World

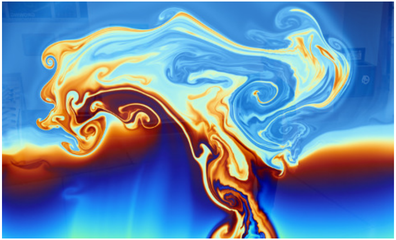
Natural Language Processing



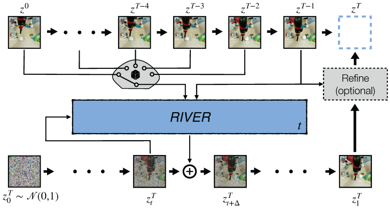
Computer Vision



Scientific Applications



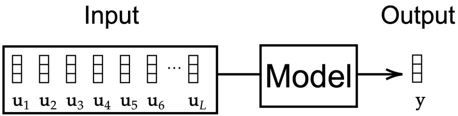
Generative AI



A Simplified Setting

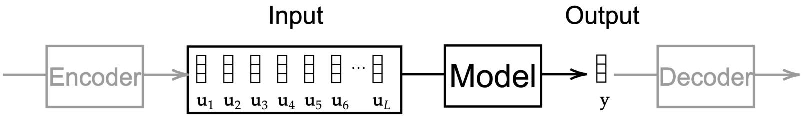
A Simplified Setting

In this talk, we observe a sequence of vectors $\mathbf{u}_1, \ldots, \mathbf{u}_L \in \mathbb{R}^m$. We want to predict an output vector $\mathbf{y} \in \mathbb{R}^p$.



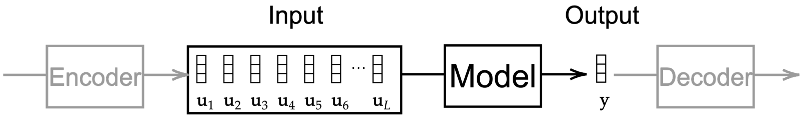
A Simplified Setting

In this talk, we observe a sequence of vectors $\mathbf{u}_1, \ldots, \mathbf{u}_L \in \mathbb{R}^m$. We want to predict an output vector $\mathbf{y} \in \mathbb{R}^p$.

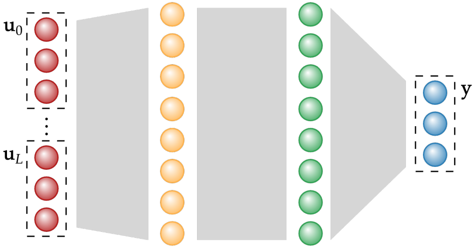


A Simplified Setting

In this talk, we observe a sequence of vectors $\mathbf{u}_1, \ldots, \mathbf{u}_L \in \mathbb{R}^m$. We want to predict an output vector $\mathbf{y} \in \mathbb{R}^p$.



Why not use a simple MLP?



Sequential Natures

Sequential Natures

1. The sequence may be long, making the MLP too large and training too inefficient.

Sequential Natures

1. The sequence may be long, making the MLP too large and training too inefficient.
2. The sequence may have varying length, making the MLP not applicable.

Sequential Natures

1. The sequence may be long, making the MLP too large and training too inefficient.
2. The sequence may have varying length, making the MLP not applicable.

The number of parameters should be independent of the sequence length L.

Sequential Natures

1. The sequence may be long, making the MLP too large and training too inefficient.
2. The sequence may have varying length, making the MLP not applicable.
3. The sequence may come in sequence, making the inference impossible until we receive the full input.

The number of parameters should be independent of the sequence length L.

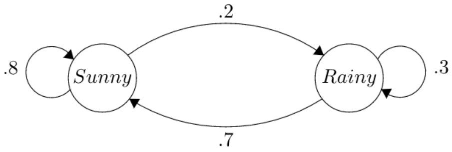
Sequential Natures

1. The sequence may be long, making the MLP too large and training too inefficient.
2. The sequence may have varying length, making the MLP not applicable.

> The number of parameters should be independent of the sequence length L.

3. The sequence may come in sequence, making the inference impossible until we receive the full input.

4. The sequence may contain temporal relationships that cannot be captured by the inductive bias of an MLP.



What Makes a Good Sequence Model?

What Makes a Good Sequence Model?

A good sequence model is one that is ...

What Makes a Good Sequence Model?

A good sequence model is one that is ...

1. Expressive and Accurate

What Makes a Good Sequence Model?

A good sequence model is one that is ...

1. Expressive and Accurate
   - Theoretical expressiveness

What Makes a Good Sequence Model?

A good sequence model is one that is ...

1. Expressive and Accurate
   - Theoretical expressiveness
   - Empirical accuracy

What Makes a Good Sequence Model?

A good sequence model is one that is ...

1. Expressive and Accurate
   - Theoretical expressiveness
   - Empirical accuracy
2. Efficient

What Makes a Good Sequence Model?

A good sequence model is one that is ...

1. Expressive and Accurate
   - Theoretical expressiveness
   - Empirical accuracy
2. Efficient
   - Time complexity

What Makes a Good Sequence Model?

A good sequence model is one that is ...

1. Expressive and Accurate
   - Theoretical expressiveness
   - Empirical accuracy
2. Efficient
   - Time complexity
   - Space complexity

What Makes a Good Sequence Model?

A good sequence model is one that is ...

1. Expressive and Accurate
   - Theoretical expressiveness
   - Empirical accuracy
2. Efficient
   - Time complexity
   - Space complexity
   - Parallelizability

What Makes a Good Sequence Model?

A good sequence model is one that is ...

1. Expressive and Accurate
   - Theoretical expressiveness
   - Empirical accuracy
2. Efficient
   - Time complexity
   - Space complexity
   - Parallelizability
3. Easy to train

What Makes a Good Sequence Model?

A good sequence model is one that is ...

1. Expressive and Accurate
   - Theoretical expressiveness
   - Empirical accuracy
2. Efficient
   - Time complexity
   - Space complexity
   - Parallelizability
3. Easy to train
   - Can we escape from a local minimum?

What Makes a Good Sequence Model?

A good sequence model is one that is ...

1. Expressive and Accurate
   - Theoretical expressiveness
   - Empirical accuracy
2. Efficient
   - Time complexity
   - Space complexity
   - Parallelizability
3. Easy to train
   - Can we escape from a local minimum?
   - Does the model always converge?

What Makes a Good Sequence Model?

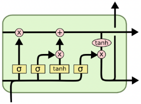
A good sequence model is one that is ...

1. Expressive and Accurate
   - Theoretical expressiveness
   - Empirical accuracy
2. Efficient
   - Time complexity
   - Space complexity
   - Parallelizability
3. Easy to train
   - Can we escape from a local minimum?
   - Does the model always converge?
4. ... (e.g., robustness to noises, multiscale modeling)

A Historical Overview

A Historical Overview

- **1980 — The Beginning**
  - RNNs
  - LSTMs
- **2014 — Golden Age of Recurrent Units**
  - GRUs
  - Seq2Seq
  - RNNsearch
  - Unitary RNNs
- **2017 — Rise of Transformers**
  - Attention is all you need
  - BERT, GPT
- **2019 — Very Long Sequences**
  - Longformer, etc.
  - State Space Models
  - Long Expressive Memory Models
- **2021 — GenAI for Sequences**
  - Chat-GPT, LLaMA, Diffusion Models, etc.



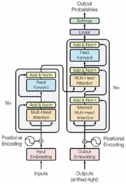


Figure 1: The Transformer - model architecture.

Recurrent Units

Recurrent Neural Networks

Recurrent Neural Networks

A recurring theme in sequential models is to keep a latent state and update it with new inputs. Recurrent neural networks (RNNs) form a most straightforward example of this idea.

Recurrent Neural Networks

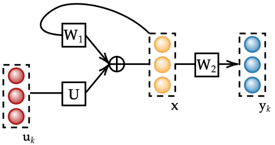
A recurring theme in sequential models is to keep a latent state and update it with new inputs. Recurrent neural networks (RNNs) form a most straightforward example of this idea.

$$\mathbf{x}_k = \tanh(\mathbf{W}_1 \mathbf{x}_{k-1} + \mathbf{U}\mathbf{u}_k + \mathbf{b}_1),$$
$$\mathbf{y}_k = \mathrm{ReLU}(\mathbf{W}_2 \mathbf{x}_k + \mathbf{b}_2).$$

Recurrent Neural Networks

A recurring theme in sequential models is to keep a latent state and update it with new inputs. Recurrent neural networks (RNNs) form a most straightforward example of this idea.

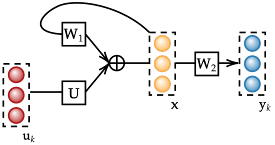
$$\mathbf{x}_k = \tanh(\mathbf{W}_1\mathbf{x}_{k-1} + \mathbf{U}\mathbf{u}_k + \mathbf{b}_1),$$
$$\mathbf{y}_k = \mathrm{ReLU}(\mathbf{W}_2\mathbf{x}_k + \mathbf{b}_2).$$

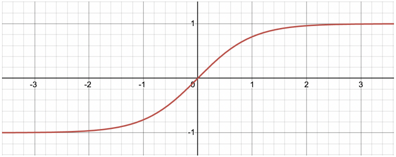


Recurrent Neural Networks

A recurring theme in sequential models is to keep a latent state and update it with new inputs. Recurrent neural networks (RNNs) form a most straightforward example of this idea.

$$\mathbf{x}_k = \tanh(\mathbf{W}_1 \mathbf{x}_{k-1} + \mathbf{U}\mathbf{u}_k + \mathbf{b}_1),$$
$$\mathbf{y}_k = \mathrm{ReLU}(\mathbf{W}_2 \mathbf{x}_k + \mathbf{b}_2).$$



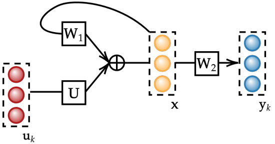


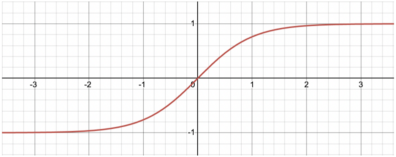
Hyperbolic Tangent: tanh

Recurrent Neural Networks

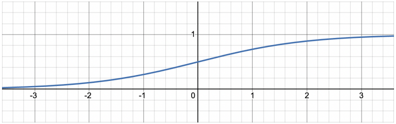
A recurring theme in sequential models is to keep a latent state and update it with new inputs. Recurrent neural networks (RNNs) form a most straightforward example of this idea.

$$\mathbf{x}_k = \tanh(\mathbf{W}_1\mathbf{x}_{k-1} + \mathbf{U}\mathbf{u}_k + \mathbf{b}_1),$$
$$\mathbf{y}_k = \mathrm{ReLU}(\mathbf{W}_2\mathbf{x}_k + \mathbf{b}_2).$$





Hyperbolic Tangent: tanh

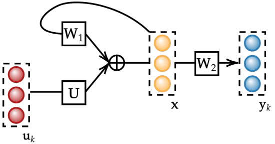


Sigmoid: σ

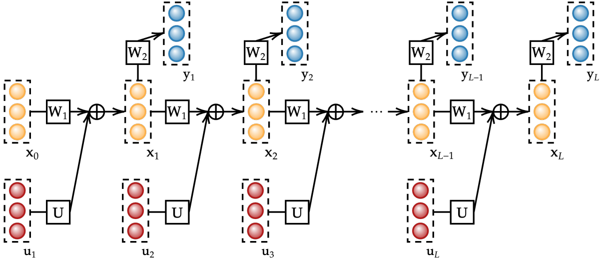
Recurrent Neural Networks

A recurring theme in sequential models is to keep a latent state and update it with new inputs. Recurrent neural networks (RNNs) form a most straightforward example of this idea.

$$\mathbf{x}_k = \tanh(\mathbf{W}_1\mathbf{x}_{k-1} + \mathbf{U}\mathbf{u}_k + \mathbf{b}_1),$$
$$\mathbf{y}_k = \mathrm{ReLU}(\mathbf{W}_2\mathbf{x}_k + \mathbf{b}_2).$$



Unrolling an RNN:



Expressiveness of RNNs

Expressiveness of RNNs

Good news: RNNs are universal approximators.

(Schäfer and Zimmermann, 2006)

Consider a finite-horizon dynamical system

$$\mathbf{x}_k = f(\mathbf{x}_{k-1}, \mathbf{u}_k),$$
$$\mathbf{y}_k = g(\mathbf{x}_k),$$

where f is measurable and g is continuous. It is arbitrarily close (in the operator sense) to an RNN with a potentially larger latent state-space dimension (i.e., the size of $\mathbf{x}$).

Expressiveness of RNNs

Good news: RNNs are universal approximators.

(Schäfer and Zimmermann, 2006)

Consider a finite-horizon dynamical system

$$\mathbf{x}_k = f(\mathbf{x}_{k-1}, \mathbf{u}_k),$$
$$\mathbf{y}_k = g(\mathbf{x}_k),$$

where f is measurable and g is continuous. It is arbitrarily close (in the operator sense) to an RNN with a potentially larger latent state-space dimension (i.e., the size of $\mathbf{x}$).

Bad news: RNNs are empirically bad at capturing long-range dependencies (LRD).

Expressiveness of RNNs

Good news: RNNs are universal approximators.

(Schäfer and Zimmermann, 2006)

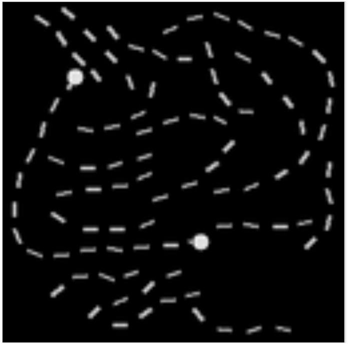
Consider a finite-horizon dynamical system

$$\mathbf{x}_k = f(\mathbf{x}_{k-1}, \mathbf{u}_k),$$
$$\mathbf{y}_k = g(\mathbf{x}_k),$$

where f is measurable and g is continuous. It is arbitrarily close (in the operator sense) to an RNN with a potentially larger latent state-space dimension (i.e., the size of $\mathbf{x}$).

Bad news: RNNs are empirically bad at capturing long-range dependencies (LRD).

Is the maze solvable?🤔



Expressiveness of RNNs

Good news: RNNs are universal approximators.

(Schäfer and Zimmermann, 2006)

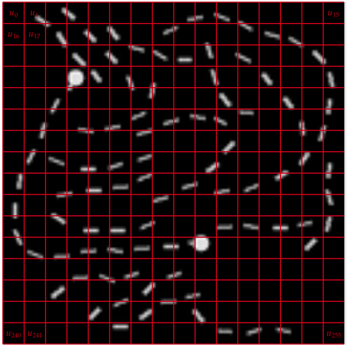
Consider a finite-horizon dynamical system

$$\mathbf{x}_k = f(\mathbf{x}_{k-1}, \mathbf{u}_k),$$
$$\mathbf{y}_k = g(\mathbf{x}_k),$$

where f is measurable and g is continuous. It is arbitrarily close (in the operator sense) to an RNN with a potentially larger latent state-space dimension (i.e., the size of $\mathbf{x}$).

Bad news: RNNs are empirically bad at capturing long-range dependencies (LRD).

Is the maze solvable?🤔



Efficiency of RNNs

Efficiency of RNNs

On a CPU...

Efficiency of RNNs

On a CPU...

- As $L \to \infty$, the computational time of the model is $\mathcal{O}(L)$.

Efficiency of RNNs

On a CPU...

- As $L \to \infty$, the computational time of the model is $\mathcal{O}(L)$.
- As $L \to \infty$, the space complexity is $\mathcal{O}(L)$ for training and $\mathcal{O}(1)$ for inferencing.

Efficiency of RNNs

On a CPU...

- As $L \to \infty$, the computational time of the model is $\mathcal{O}(L)$.
- As $L \to \infty$, the space complexity is $\mathcal{O}(L)$ for training and $\mathcal{O}(1)$ for inferencing.

On a GPU...

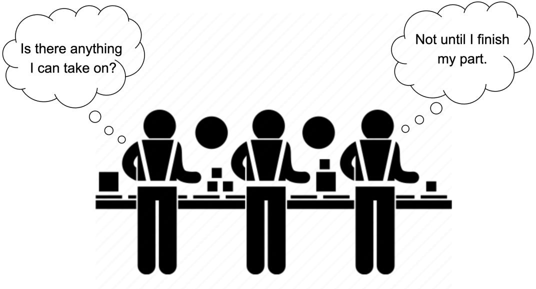
Efficiency of RNNs

On a CPU...

- As $L \to \infty$, the computational time of the model is $\mathcal{O}(L)$.
- As $L \to \infty$, the space complexity is $\mathcal{O}(L)$ for training and $\mathcal{O}(1)$ for inferencing.

On a GPU...

- The gradient has to be computed recurrently. Hence, no parallelization can be done along the time axis. In particular, it takes $\mathcal{O}(L \cdot \text{time per step})$ even on a GPU.



Training Stability of RNNs

Training Stability of RNNs

RNNs are not stable over training. They suffer from the infamous vanishing and exploding gradient issues.

Training Stability of RNNs

RNNs are not stable over training. They suffer from the infamous vanishing and exploding gradient issues.

Gradients of a Linear RNN

Consider a simplified linear RNN with no bias term: $\mathbf{x}_k = \mathbf{W}\mathbf{x}_{k-1} + \mathbf{U}\mathbf{u}_k$. Given a generic loss function $\mathcal{L}$, the gradient is

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}} = \sum_{k=1}^{L} \frac{\partial \mathcal{L}}{\partial \mathbf{x}_k} \frac{\partial \mathbf{x}_k}{\partial \mathbf{W}} = \sum_{k=1}^{L} \left(\frac{\partial \mathcal{L}}{\partial \mathbf{x}_k} \sum_{j=1}^{k-1} \mathbf{W}^j \mathbf{x}_{k-j} \right).$$

Training Stability of RNNs

RNNs are not stable over training. They suffer from the infamous vanishing and exploding gradient issues.

Gradients of a Linear RNN

Consider a simplified linear RNN with no bias term: $\mathbf{x}_k = \mathbf{W}\mathbf{x}_{k-1} + \mathbf{U}\mathbf{u}_k$. Given a generic loss function $\mathcal{L}$, the gradient is

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}} = \sum_{k=1}^{L} \frac{\partial \mathcal{L}}{\partial \mathbf{x}_k} \frac{\partial \mathbf{x}_k}{\partial \mathbf{W}} = \sum_{k=1}^{L} \left(\frac{\partial \mathcal{L}}{\partial \mathbf{x}_k} \sum_{j=1}^{k-1} \mathbf{W}^j \mathbf{x}_{k-j} \right).$$

If $\rho(\mathbf{W}) > 1$, then $\|\mathbf{W}^j\|_2$ explodes exponentially as $j \to \infty$.

Training Stability of RNNs

RNNs are not stable over training. They suffer from the infamous vanishing and exploding gradient issues.

Gradients of a Linear RNN

Consider a simplified linear RNN with no bias term: $\mathbf{x}_k = \mathbf{W}\mathbf{x}_{k-1} + \mathbf{U}\mathbf{u}_k$. Given a generic loss function $\mathcal{L}$, the gradient is

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}} = \sum_{k=1}^{L} \frac{\partial \mathcal{L}}{\partial \mathbf{x}_k} \frac{\partial \mathbf{x}_k}{\partial \mathbf{W}} = \sum_{k=1}^{L} \left(\frac{\partial \mathcal{L}}{\partial \mathbf{x}_k} \sum_{j=1}^{k-1} \mathbf{W}^j \mathbf{x}_{k-j} \right).$$

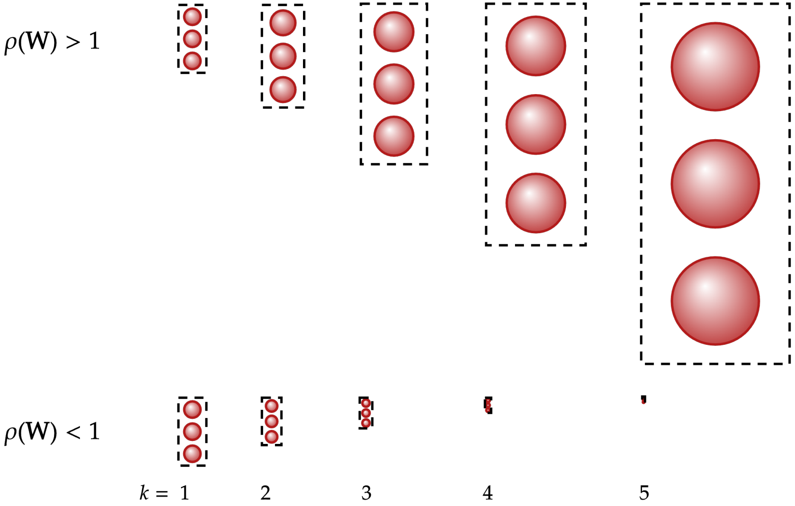
If $\rho(\mathbf{W}) > 1$, then $\|\mathbf{W}^j\|_2$ explodes exponentially as $j \to \infty$.

If $\rho(\mathbf{W}) < 1$, then $\|\mathbf{W}^j\|_2$ vanishes exponentially as $j \to \infty$.

Why Do We Observe Vanishing/Exploding Gradients?

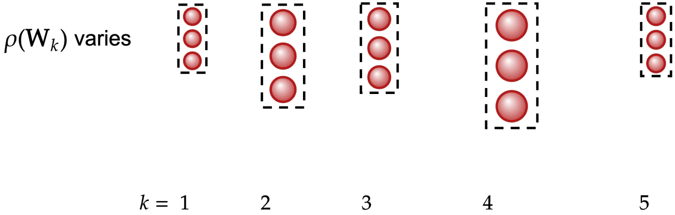
Why Do We Observe Vanishing/Exploding Gradients?

The memory of an input is dampened or magnified by a constant factor.



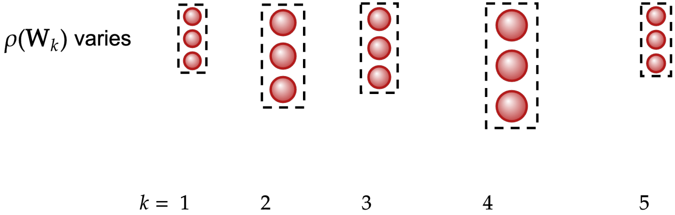
Why Do We Observe Vanishing/Exploding Gradients?

If we can make the memory decay or amplify differently at every step, then we can reduce the vanishing/exploding gradient issues.



Why Do We Observe Vanishing/Exploding Gradients?

If we can make the memory decay or amplify differently at every step, then we can reduce the vanishing/exploding gradient issues.



This is partially why a deep MLP does not suffer from such issues. Unfortunately, we cannot train a different $\mathbf{W}_k$ for each step k. We need to be smarter in constructing the recurrent unit.

Long Short-Term Memory

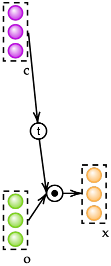
Long Short-Term Memory

Long short-term memory (LSTM) [Hochreiter and Schmidhuber, 1997] is a variant of an SSM that incorporates a long-term memory cell.

Long Short-Term Memory

Long short-term memory (LSTM) [Hochreiter and Schmidhuber, 1997] is a variant of an SSM that incorporates a long-term memory cell.

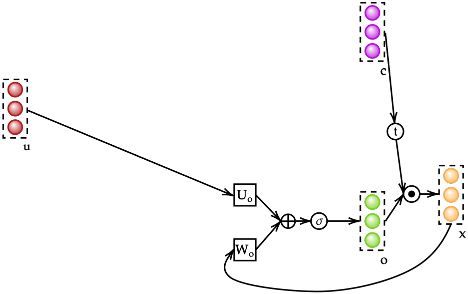
$$\mathbf{x}_k = \mathbf{o}_k \circ \tanh(\mathbf{c}_k),$$



Long Short-Term Memory

Long short-term memory (LSTM) [Hochreiter and Schmidhuber, 1997] is a variant of an SSM that incorporates a long-term memory cell.

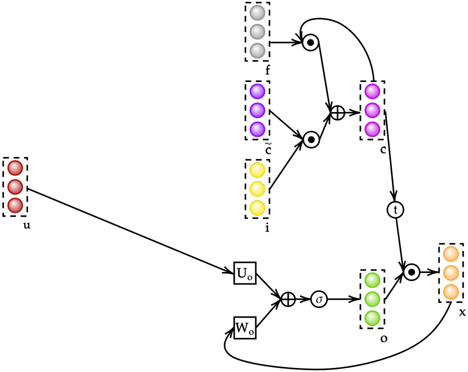
$$\mathbf{x}_k = \mathbf{o}_k \circ \tanh(\mathbf{c}_k),$$
$$\mathbf{o}_k = \sigma(\mathbf{W_o}\mathbf{x}_{k-1} + \mathbf{U_o}\mathbf{u}_k + \mathbf{b_o}),$$



Long Short-Term Memory

Long short-term memory (LSTM) [Hochreiter and Schmidhuber, 1997] is a variant of an SSM that incorporates a long-term memory cell.

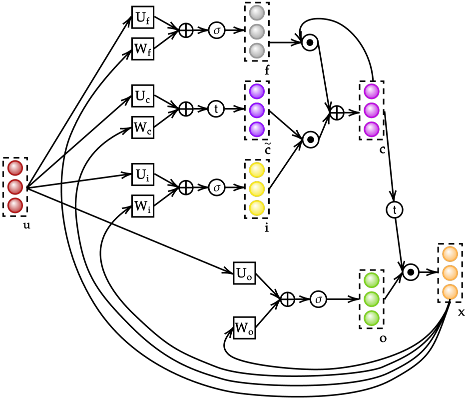
$$
\begin{aligned}
\mathbf{x}_k &= \mathbf{o}_k \circ \tanh(\mathbf{c}_k),\\
\mathbf{o}_k &= \sigma(\mathbf{W_o}\mathbf{x}_{k-1} + \mathbf{U_o}\mathbf{u}_k + \mathbf{b_o}),\\
\mathbf{c}_k &= \mathbf{f}_k \circ \mathbf{c}_{k-1} + \mathbf{i}_k \circ \tilde{\mathbf{c}}_k,
\end{aligned}
$$



Long Short-Term Memory

Long short-term memory (LSTM) [Hochreiter and Schmidhuber, 1997] is a variant of an SSM that incorporates a long-term memory cell.

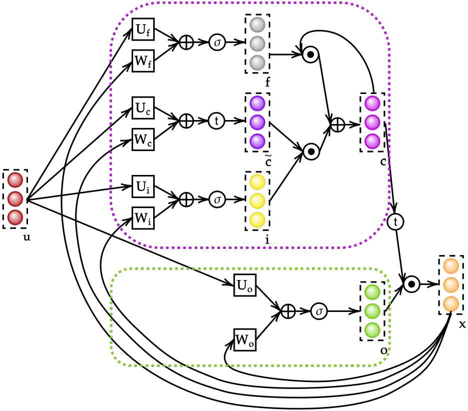
$$
\begin{aligned}
\mathbf{x}_k &= \mathbf{o}_k \circ \tanh(\mathbf{c}_k), \\
\mathbf{o}_k &= \sigma(\mathbf{W_o}\mathbf{x}_{k-1} + \mathbf{U_o}\mathbf{u}_k + \mathbf{b_o}), \\
\mathbf{c}_k &= \mathbf{f}_k \circ \mathbf{c}_{k-1} + \mathbf{i}_k \circ \tilde{\mathbf{c}}_k, \\
\mathbf{f}_k &= \sigma(\mathbf{W_f}\mathbf{x}_{k-1} + \mathbf{U_f}\mathbf{u}_k + \mathbf{b_f}), \\
\mathbf{i}_k &= \sigma(\mathbf{W_i}\mathbf{x}_{k-1} + \mathbf{U_i}\mathbf{u}_k + \mathbf{b_i}), \\
\tilde{\mathbf{c}}_k &= \tanh(\mathbf{W_c}\mathbf{x}_{k-1} + \mathbf{U_c}\mathbf{u}_k + \mathbf{b_c}).
\end{aligned}
$$



Long Short-Term Memory

Long short-term memory (LSTM) [Hochreiter and Schmidhuber, 1997] is a variant of an SSM that incorporates a long-term memory cell.

$$
\begin{aligned}
\mathbf{x}_k &= \mathbf{o}_k \circ \tanh(\mathbf{c}_k),\\
\mathbf{o}_k &= \sigma(\mathbf{W_o}\mathbf{x}_{k-1} + \mathbf{U_o}\mathbf{u}_k + \mathbf{b_o}),\\
\mathbf{c}_k &= \mathbf{f}_k \circ \mathbf{c}_{k-1} + \mathbf{i}_k \circ \tilde{\mathbf{c}}_k,\\
\mathbf{f}_k &= \sigma(\mathbf{W_f}\mathbf{x}_{k-1} + \mathbf{U_f}\mathbf{u}_k + \mathbf{b_f}),\\
\mathbf{i}_k &= \sigma(\mathbf{W_i}\mathbf{x}_{k-1} + \mathbf{U_i}\mathbf{u}_k + \mathbf{b_i}),\\
\tilde{\mathbf{c}}_k &= \tanh(\mathbf{W_c}\mathbf{x}_{k-1} + \mathbf{U_c}\mathbf{u}_k + \mathbf{b_c}).
\end{aligned}
$$



Gated Recurrent Unit

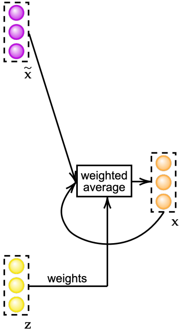
Gated Recurrent Unit

Gated recurrent units (GRUs) [Cho et al., 2014] are similar to LSTMs in many sense.

Gated Recurrent Unit

Gated recurrent units (GRUs) [Cho et al., 2014] are similar to LSTMs in many sense.

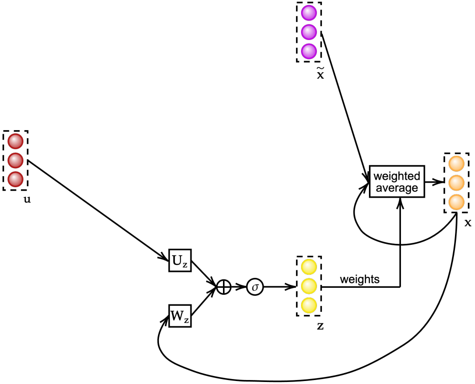
$$\mathbf{x}_k = (1 - \mathbf{z}_k) \circ \mathbf{x}_{k-1} + \mathbf{z}_k \circ \tilde{\mathbf{x}}_k,$$



Gated Recurrent Unit

Gated recurrent units (GRUs) [Cho et al., 2014] are similar to LSTMs in many sense.

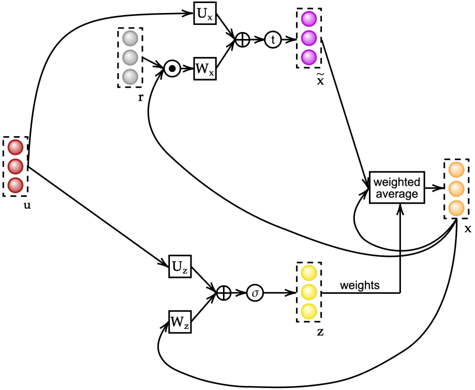
$$\mathbf{x}_k = (1 - \mathbf{z}_k) \circ \mathbf{x}_{k-1} + \mathbf{z}_k \circ \tilde{\mathbf{x}}_k,$$
$$\mathbf{z}_k = \sigma(\mathbf{W}_z \mathbf{x}_{k-1} + \mathbf{U}_z \mathbf{u}_k + \mathbf{b}_z),$$



Gated Recurrent Unit

Gated recurrent units (GRUs) [Cho et al., 2014] are similar to LSTMs in many sense.

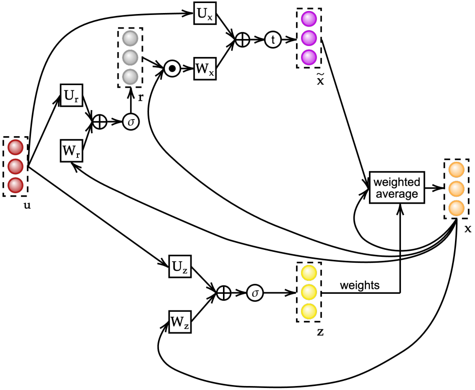
$$
\begin{aligned}
\mathbf{x}_k &= (1 - \mathbf{z}_k) \circ \mathbf{x}_{k-1} + \mathbf{z}_k \circ \tilde{\mathbf{x}}_k, \\
\mathbf{z}_k &= \sigma(\mathbf{W}_\mathbf{z}\mathbf{x}_{k-1} + \mathbf{U}_\mathbf{z}\mathbf{u}_k + \mathbf{b}_\mathbf{z}), \\
\tilde{\mathbf{x}}_k &= \tanh(\mathbf{W}_\mathbf{x}(\mathbf{r}_k \circ \mathbf{x}_{k-1}) + \mathbf{U}_\mathbf{x}\mathbf{u}_k + \mathbf{b}_\mathbf{h}),
\end{aligned}
$$



Gated Recurrent Unit

Gated recurrent units (GRUs) [Cho et al., 2014] are similar to LSTMs in many sense.

$$
\begin{aligned}
\mathbf{x}_k &= (1 - \mathbf{z}_k) \circ \mathbf{x}_{k-1} + \mathbf{z}_k \circ \tilde{\mathbf{x}}_k, \\
\mathbf{z}_k &= \sigma(\mathbf{W}_\mathbf{z}\mathbf{x}_{k-1} + \mathbf{U}_\mathbf{z}\mathbf{u}_k + \mathbf{b}_\mathbf{z}), \\
\tilde{\mathbf{x}}_k &= \tanh(\mathbf{W}_\mathbf{x}(\mathbf{r}_k \circ \mathbf{x}_{k-1}) + \mathbf{U}_\mathbf{x}\mathbf{u}_k + \mathbf{b}_\mathbf{h}), \\
\mathbf{r}_k &= \sigma(\mathbf{W}_\mathbf{r}\mathbf{x}_{k-1} + \mathbf{U}_\mathbf{r}\mathbf{u}_k + \mathbf{b}_\mathbf{r}),
\end{aligned}
$$



Properties of LSTMs and GRUs

Properties of LSTMs and GRUs

LSTMs and GRUs are ...

Properties of LSTMs and GRUs

LSTMs and GRUs are ...

1. Are universal approximators.

Properties of LSTMs and GRUs

LSTMs and GRUs are ...

1. Are universal approximators.
2. Share the same time and space complexities with RNNs.

Properties of LSTMs and GRUs

LSTMs and GRUs are ...

1. Are universal approximators.
2. Share the same time and space complexities with RNNs.
3. Suffer less from the vanishing or exploding gradient issues.

Properties of LSTMs and GRUs

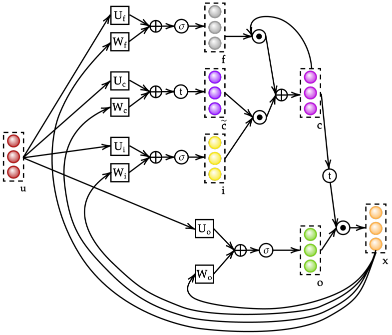
LSTMs and GRUs are ...

1. Are universal approximators.
2. Share the same time and space complexities with RNNs.
3. Suffer less from the vanishing or exploding gradient issues.
   - Key idea: the memory decay/enhancement is not constant per step.

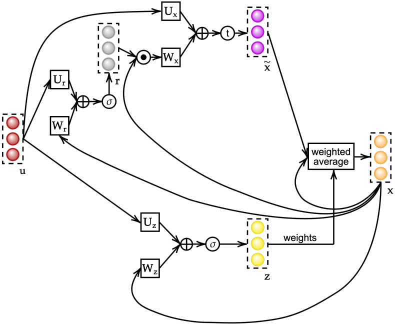
Properties of LSTMs and GRUs

LSTMs and GRUs are ...

1. Are universal approximators.
2. Share the same time and space complexities with RNNs.
3. Suffer less from the vanishing or exploding gradient issues.
    - Key idea: the memory decay/enhancement is not constant per step.



LSTM



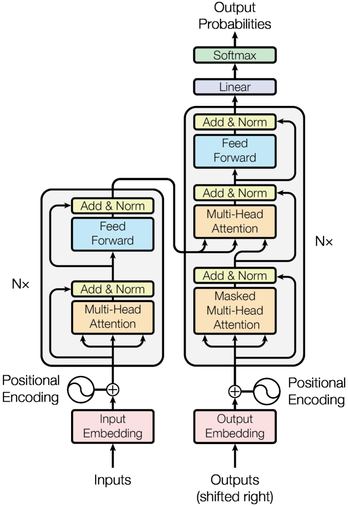
GRU

Other Sequential Models

Transformers

Transformers

Transformers form a class of models that are wildly used in NLP and CV.



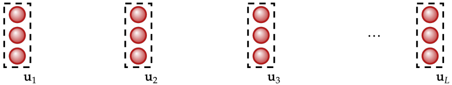
[Vaswani et al., 2017]

Transformers

The backbone of a transformer is called the attention mechanism. Compared to recurrent models, attention explicitly seeks a connection between every pair of elements in a sequence.

Transformers

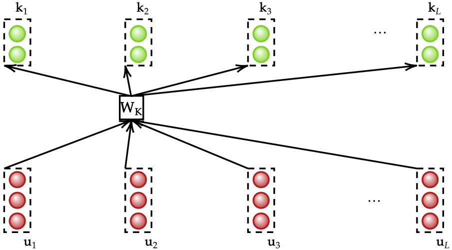
The backbone of a transformer is called the attention mechanism. Compared to recurrent models, attention explicitly seeks a connection between every pair of elements in a sequence.



Transformers

The backbone of a transformer is called the attention mechanism. Compared to recurrent models, attention explicitly seeks a connection between every pair of elements in a sequence.

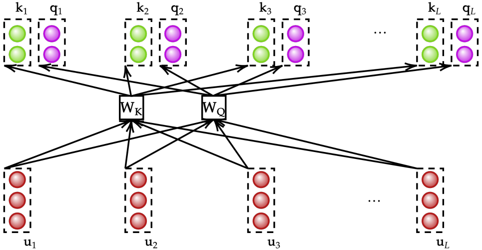
$\mathbf{k}_i = \mathbf{W_k x}_i,$



Transformers

The backbone of a transformer is called the attention mechanism. Compared to recurrent models, attention explicitly seeks a connection between every pair of elements in a sequence.

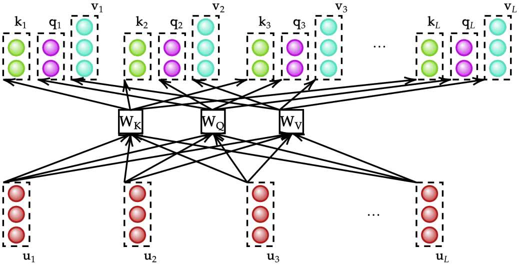
$$\mathbf{k}_i = \mathbf{W}_\mathbf{k}\mathbf{x}_i,$$
$$\mathbf{q}_i = \mathbf{W}_\mathbf{q}\mathbf{x}_i,$$



Transformers

The backbone of a transformer is called the attention mechanism. Compared to recurrent models, attention explicitly seeks a connection between every pair of elements in a sequence.

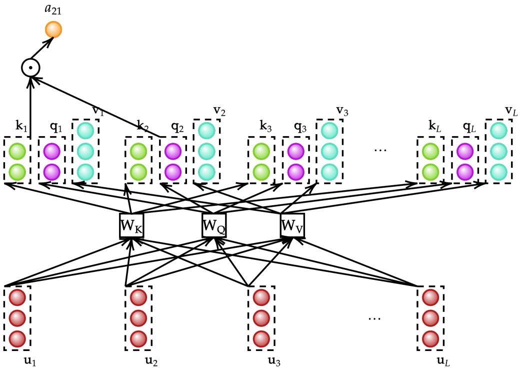
$$\mathbf{k}_i = \mathbf{W_k x}_i,$$
$$\mathbf{q}_i = \mathbf{W_q x}_i,$$
$$\mathbf{v}_i = \mathbf{W_v x}_i,$$



Transformers

The backbone of a transformer is called the attention mechanism. Compared to recurrent models, attention explicitly seeks a connection between every pair of elements in a sequence.

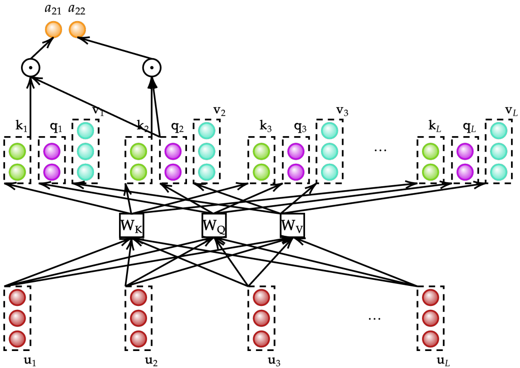
$$\mathbf{k}_i = \mathbf{W}_\mathbf{k}\mathbf{x}_i,$$
$$\mathbf{q}_i = \mathbf{W}_\mathbf{q}\mathbf{x}_i,$$
$$\mathbf{v}_i = \mathbf{W}_\mathbf{v}\mathbf{x}_i,$$
$$a_{ij} = \mathbf{q}_i^\top \mathbf{k}_j,$$



Transformers

The backbone of a transformer is called the attention mechanism. Compared to recurrent models, attention explicitly seeks a connection between every pair of elements in a sequence.

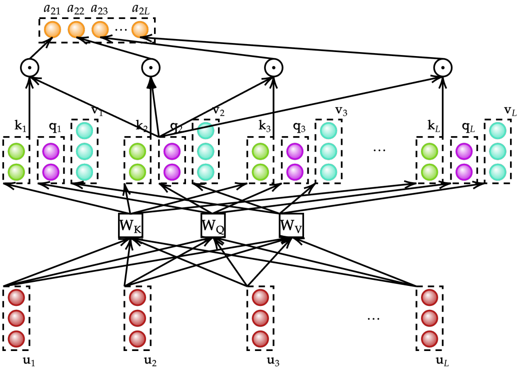
$$\mathbf{k}_i = \mathbf{W}_\mathbf{k}\mathbf{x}_i,$$
$$\mathbf{q}_i = \mathbf{W}_\mathbf{q}\mathbf{x}_i,$$
$$\mathbf{v}_i = \mathbf{W}_\mathbf{v}\mathbf{x}_i,$$
$$a_{ij} = \mathbf{q}_i^\top \mathbf{k}_j,$$



Transformers

The backbone of a transformer is called the attention mechanism. Compared to recurrent models, attention explicitly seeks a connection between every pair of elements in a sequence.

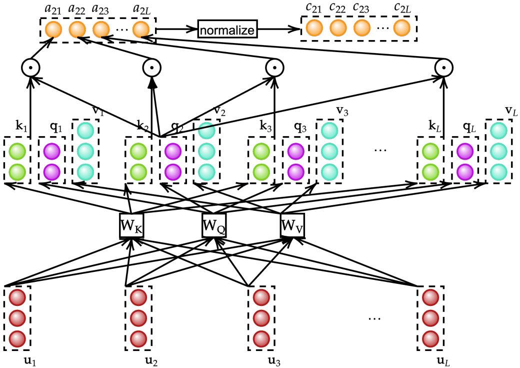
$$\mathbf{k}_i = \mathbf{W}_\mathbf{k}\mathbf{x}_i,$$
$$\mathbf{q}_i = \mathbf{W}_\mathbf{q}\mathbf{x}_i,$$
$$\mathbf{v}_i = \mathbf{W}_\mathbf{v}\mathbf{x}_i,$$
$$a_{ij} = \mathbf{q}_i^\top \mathbf{k}_j,$$



Transformers

The backbone of a transformer is called the attention mechanism. Compared to recurrent models, attention explicitly seeks a connection between every pair of elements in a sequence.

$$\mathbf{k}_i = \mathbf{W}_\mathbf{k}\mathbf{x}_i,$$
$$\mathbf{q}_i = \mathbf{W}_\mathbf{q}\mathbf{x}_i,$$
$$\mathbf{v}_i = \mathbf{W}_\mathbf{v}\mathbf{x}_i,$$
$$a_{ij} = \mathbf{q}_i^\top \mathbf{k}_j,$$
$$c_{ij} = \frac{\exp(a_{ij}/\sqrt{d})}{\sum_{j=1}^{L}\exp(a_{ij}/\sqrt{d})},$$



Transformers

The backbone of a transformer is called the attention mechanism. Compared to recurrent models, attention explicitly seeks a connection between every pair of elements in a sequence.

$$\mathbf{k}_i = \mathbf{W}_\mathbf{k}\mathbf{x}_i,$$

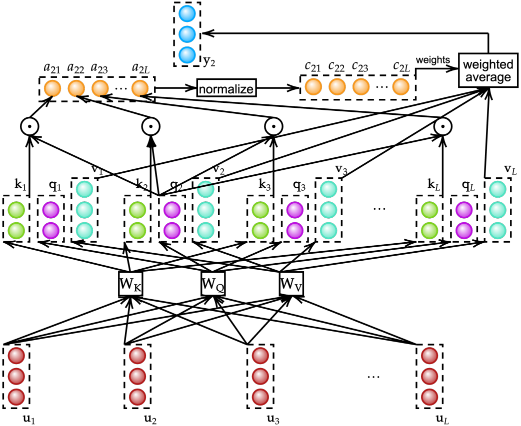
$$\mathbf{q}_i = \mathbf{W}_\mathbf{q}\mathbf{x}_i,$$

$$\mathbf{v}_i = \mathbf{W}_\mathbf{v}\mathbf{x}_i,$$

$$a_{ij} = \mathbf{q}_i^\top \mathbf{k}_j,$$

$$c_{ij} = \frac{\exp(a_{ij}/\sqrt{d})}{\sum_{j=1}^{L}\exp(a_{ij}/\sqrt{d})},$$

$$\mathbf{y}_i = \sum_{j=1}^{L} c_{ij}\mathbf{v}_j.$$



Properties of Transformers

Properties of Transformers

- Every element in the sequence is in a symmetric position. There is no natural inductive bias over the time axis.

Properties of Transformers

- Every element in the sequence is in a symmetric position. There is no natural inductive bias over the time axis.
- Without any parallelization, computing the attention takes $\mathcal{O}(L^2)$ as $L \to \infty$.

$$\mathbf{k}_i = \mathbf{W_k}\mathbf{x}_i,\, \mathbf{q}_i = \mathbf{W_q}\mathbf{x}_i,$$

$$\mathbf{v}_i = \mathbf{W_v}\mathbf{x}_i,\, a_{ij} = \mathbf{q}_i^\top \mathbf{k}_j,$$

$$c_{ij} = \frac{\exp(a_{ij}/\sqrt{d})}{\sum_{j=1}^{L}\exp(a_{ij}/\sqrt{d})},$$

$$\mathbf{y}_i = \sum_{j=1}^{L} c_{ij}\mathbf{v}_j.$$

Properties of Transformers

- Every element in the sequence is in a symmetric position. There is no natural inductive bias over the time axis.
- Without any parallelization, computing the attention takes $\mathcal{O}(L^2)$ as $L \to \infty$.
- However, it is very parallelizable ...

$$\mathbf{k}_i = \mathbf{W}_\mathbf{k}\mathbf{x}_i, \mathbf{q}_i = \mathbf{W}_\mathbf{q}\mathbf{x}_i,$$

$$\mathbf{v}_i = \mathbf{W}_\mathbf{v}\mathbf{x}_i, a_{ij} = \mathbf{q}_i^\top \mathbf{k}_j,$$

$$c_{ij} = \frac{\exp(a_{ij}/\sqrt{d})}{\sum_{j=1}^{L} \exp(a_{ij}/\sqrt{d})},$$

$$\mathbf{y}_i = \sum_{j=1}^{L} c_{ij}\mathbf{v}_j.$$

Properties of Transformers

- Every element in the sequence is in a symmetric position. There is no natural inductive bias over the time axis.
- Without any parallelization, computing the attention takes $\mathcal{O}(L^2)$ as $L \to \infty$.
- However, it is very parallelizable ...
  - for inferencing;

$$\mathbf{k}_i = \mathbf{W}_\mathbf{k}\mathbf{x}_i, \mathbf{q}_i = \mathbf{W}_\mathbf{q}\mathbf{x}_i,$$

$$\mathbf{v}_i = \mathbf{W}_\mathbf{v}\mathbf{x}_i, a_{ij} = \mathbf{q}_i^\top \mathbf{k}_j,$$

$$c_{ij} = \frac{\exp(a_{ij}/\sqrt{d})}{\sum_{j=1}^{L} \exp(a_{ij}/\sqrt{d})},$$

$$\mathbf{y}_i = \sum_{j=1}^{L} c_{ij}\mathbf{v}_j.$$

Properties of Transformers

- Every element in the sequence is in a symmetric position. There is no natural inductive bias over the time axis.
- Without any parallelization, computing the attention takes $\mathcal{O}(L^2)$ as $L \to \infty$.
- However, it is very parallelizable ...
  - for inferencing;
  - for backpropagation, but the softmax raises some difficulties in parallelization.

$$\mathbf{k}_i = \mathbf{W}_\mathbf{k}\mathbf{x}_i, \mathbf{q}_i = \mathbf{W}_\mathbf{q}\mathbf{x}_i,$$

$$\mathbf{v}_i = \mathbf{W}_\mathbf{v}\mathbf{x}_i, a_{ij} = \mathbf{q}_i^\top \mathbf{k}_j,$$

$$c_{ij} = \frac{\exp(a_{ij}/\sqrt{d})}{\sum_{j=1}^{L} \exp(a_{ij}/\sqrt{d})},$$

$$\mathbf{y}_i = \sum_{j=1}^{L} c_{ij}\mathbf{v}_j.$$

Properties of Transformers

- Every element in the sequence is in a symmetric position. There is no natural inductive bias over the time axis.
- Without any parallelization, computing the attention takes $\mathcal{O}(L^2)$ as $L \to \infty$.
- However, it is very parallelizable ...
  - for inferencing;
  - for backpropagation, but the softmax raises some difficulties in parallelization.
  - Check out linear attention [Katharopoulos et al., 2020] and FlashAttention [Dao et al., 2022]!

$$\mathbf{k}_i = \mathbf{W}_\mathbf{k}\mathbf{x}_i, \mathbf{q}_i = \mathbf{W}_\mathbf{q}\mathbf{x}_i,$$

$$\mathbf{v}_i = \mathbf{W}_\mathbf{v}\mathbf{x}_i, a_{ij} = \mathbf{q}_i^\top \mathbf{k}_j,$$

$$c_{ij} = \frac{\exp(a_{ij}/\sqrt{d})}{\sum_{j=1}^{L} \exp(a_{ij}/\sqrt{d})},$$

$$\mathbf{y}_i = \sum_{j=1}^{L} c_{ij}\mathbf{v}_j.$$

Properties of Transformers

- Every element in the sequence is in a symmetric position. There is no natural inductive bias over the time axis.
- Without any parallelization, computing the attention takes $\mathcal{O}(L^2)$ as $L \to \infty$.
- However, it is very parallelizable ...
  - for inferencing;
  - for backpropagation, but the softmax raises some difficulties in parallelization.
  - Check out linear attention [Katharopoulos et al., 2020] and FlashAttention [Dao et al., 2022]!
- The model is not causal, so one cannot evaluate it without the entire sequence.

$$\mathbf{k}_i = \mathbf{W}_\mathbf{k}\mathbf{x}_i, \mathbf{q}_i = \mathbf{W}_\mathbf{q}\mathbf{x}_i,$$

$$\mathbf{v}_i = \mathbf{W}_\mathbf{v}\mathbf{x}_i, a_{ij} = \mathbf{q}_i^\top \mathbf{k}_j,$$

$$c_{ij} = \frac{\exp(a_{ij}/\sqrt{d})}{\sum_{j=1}^{L}\exp(a_{ij}/\sqrt{d})},$$

$$\mathbf{y}_i = \sum_{j=1}^{L} c_{ij}\mathbf{v}_j.$$

Properties of Transformers

- Every element in the sequence is in a symmetric position. There is no natural inductive bias over the time axis.
- Without any parallelization, computing the attention takes $\mathcal{O}(L^2)$ as $L \to \infty$.
- However, it is very parallelizable ...
  - for inferencing;
  - for backpropagation, but the softmax raises some difficulties in parallelization.
  - Check out linear attention [Katharopoulos et al., 2020] and FlashAttention [Dao et al., 2022]!

$$\mathbf{k}_i = \mathbf{W_k}\mathbf{x}_i, \mathbf{q}_i = \mathbf{W_q}\mathbf{x}_i,$$

$$\mathbf{v}_i = \mathbf{W_v}\mathbf{x}_i, a_{ij} = \mathbf{q}_i^\top \mathbf{k}_j,$$

$$c_{ij} = \frac{\exp(a_{ij}/\sqrt{d})}{\sum_{j=1}^{L} \exp(a_{ij}/\sqrt{d})},$$

$$\mathbf{y}_i = \sum_{j=1}^{L} c_{ij}\mathbf{v}_j.$$

- The model is not causal, so one cannot evaluate it without the entire sequence.
  - Check out masking!

State-Space Models

State-Space Models

A state space model (SSM) [Gu et al., 2022] is very similar to an RNN. Its recurrent units are based on linear, time-invariant (LTI) systems

$$\begin{aligned} \mathbf{x}'(t) &= \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t), \\ \mathbf{y}(t) &= \mathbf{C}\mathbf{x}(t) + \mathbf{D}\mathbf{u}(t). \end{aligned}$$

State-Space Models

A state space model (SSM) [Gu et al., 2022] is very similar to an RNN. Its recurrent units are based on linear, time-invariant (LTI) systems

$$\mathbf{x}'(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t),$$
$$\mathbf{y}(t) = \mathbf{C}\mathbf{x}(t) + \mathbf{D}\mathbf{u}(t).$$

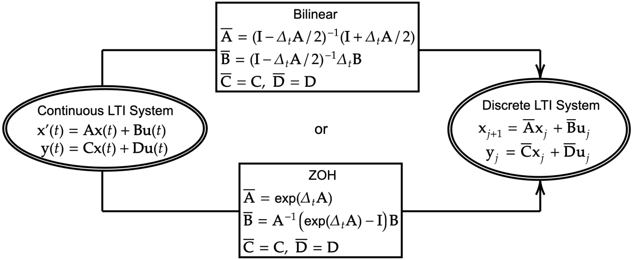
Wait... but your sequence is discrete.

State-Space Models

A state space model (SSM) [Gu et al., 2022] is very similar to an RNN. Its recurrent units are based on linear, time-invariant (LTI) systems

$$\mathbf{x}'(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t),$$
$$\mathbf{y}(t) = \mathbf{C}\mathbf{x}(t) + \mathbf{D}\mathbf{u}(t).$$

Wait... but your sequence is discrete.
We have to discretize the system with respect to some trainable sampling period $\Delta t > 0$:



SSMs vs RNNs

SSMs vs RNNs

RNN

$$\mathbf{x}_k = \tanh(\mathbf{W}_1 \mathbf{x}_{k-1} + \mathbf{U}\mathbf{u}_k + \mathbf{b}_1)$$
$$\mathbf{y}_k = \mathrm{ReLU}(\mathbf{W}_2 \mathbf{x}_k + \mathbf{b}_2)$$

SSMs vs RNNs

RNN

$$\mathbf{x}_k = \tanh(\mathbf{W}_1 \mathbf{x}_{k-1} + \mathbf{U}\mathbf{u}_k + \mathbf{b}_1)$$
$$\mathbf{y}_k = \mathrm{ReLU}(\mathbf{W}_2 \mathbf{x}_k + \mathbf{b}_2)$$

SSM

$$\mathbf{x}'(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t)$$
$$\mathbf{y}(t) = \mathbf{C}\mathbf{x}(t) + \mathbf{D}\mathbf{u}(t)$$

or

$$\mathbf{x}_k = \overline{\mathbf{A}}\mathbf{x}_{k-1} + \overline{\mathbf{B}}\mathbf{u}_k$$
$$\mathbf{y}_k = \overline{\mathbf{C}}\mathbf{x}_k + \overline{\mathbf{D}}\mathbf{u}_k$$

SSMs vs RNNs

RNN

$$\mathbf{x}_k = \tanh(\mathbf{W}_1\mathbf{x}_{k-1} + \mathbf{U}\mathbf{u}_k + \mathbf{b}_1)$$
$$\mathbf{y}_k = \mathrm{ReLU}(\mathbf{W}_2\mathbf{x}_k + \mathbf{b}_2)$$

SSM

$$\mathbf{x}'(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t)$$
$$\mathbf{y}(t) = \mathbf{C}\mathbf{x}(t) + \mathbf{D}\mathbf{u}(t)$$

or

$$\mathbf{x}_k = \overline{\mathbf{A}}\mathbf{x}_{k-1} + \overline{\mathbf{B}}\mathbf{u}_k$$
$$\mathbf{y}_k = \overline{\mathbf{C}}\mathbf{x}_k + \overline{\mathbf{D}}\mathbf{u}_k$$

SSMs vs RNNs

RNN

$$\mathbf{x}_k = \tanh(\mathbf{W}_1 \mathbf{x}_{k-1} + \mathbf{U}\mathbf{u}_k + \mathbf{b}_1)$$
$$\mathbf{y}_k = \mathrm{ReLU}(\mathbf{W}_2 \mathbf{x}_k + \mathbf{b}_2)$$

SSM

$$\mathbf{x}'(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t)$$
$$\mathbf{y}(t) = \mathbf{C}\mathbf{x}(t) + \mathbf{D}\mathbf{u}(t)$$

or

$$\mathbf{x}_k = \overline{\mathbf{A}}\mathbf{x}_{k-1} + \overline{\mathbf{B}}\mathbf{u}_k$$
$$\mathbf{y}_k = \overline{\mathbf{C}}\mathbf{x}_k + \overline{\mathbf{D}}\mathbf{u}_k$$

What are the main differences between an RNN and an SSM?

1. An RNN is nonlinear while an SSM is linear.
2. An RNN is completely discrete while an SSM has an underlying continuous system.

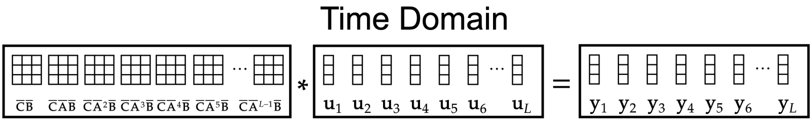
Efficiency of SSMs

Efficiency of SSMs

An LTI system is linear. Hence, it can be evaluated more easily.

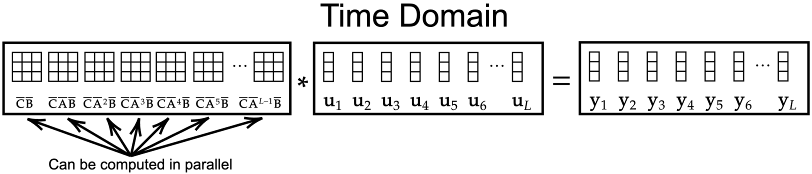
Efficiency of SSMs

An LTI system is linear. Hence, it can be evaluated more easily.



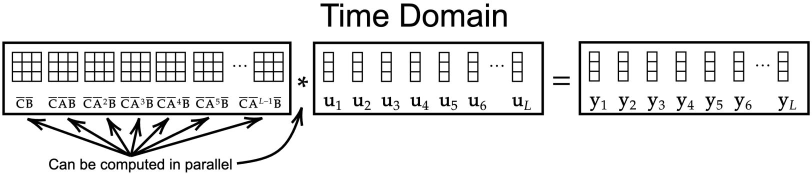
Efficiency of SSMs

An LTI system is linear. Hence, it can be evaluated more easily.



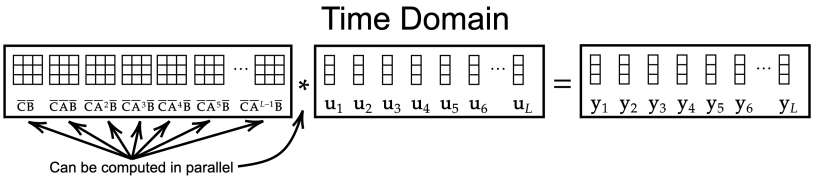
Efficiency of SSMs

An LTI system is linear. Hence, it can be evaluated more easily.



Efficiency of SSMs

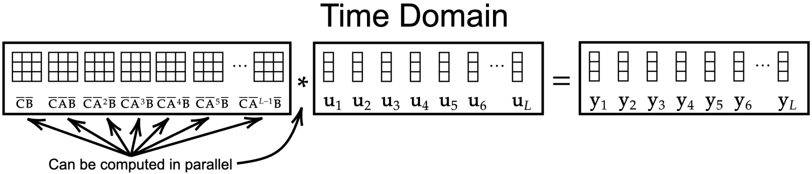
An LTI system is linear. Hence, it can be evaluated more easily.



Assume we have L processors that can be run in parallel.

Efficiency of SSMs

An LTI system is linear. Hence, it can be evaluated more easily.

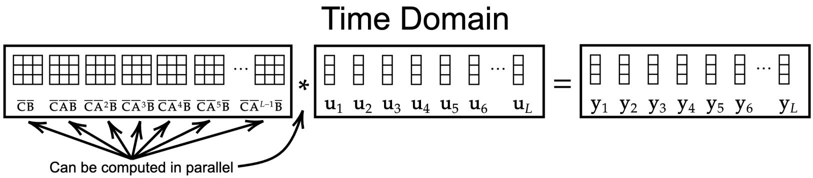


Assume we have L processors that can be run in parallel.

- Time complexity of RNN: $\mathcal{O}(L \cdot$ time per step).

Efficiency of SSMs

An LTI system is linear. Hence, it can be evaluated more easily.

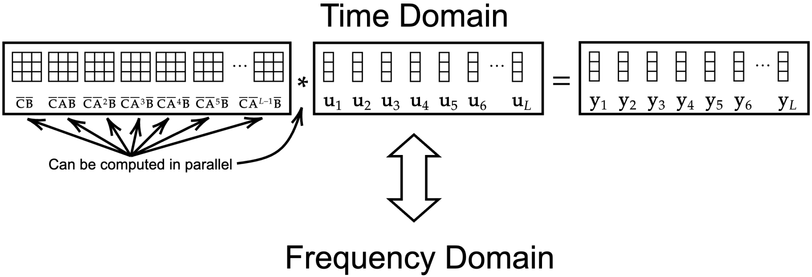


Assume we have L processors that can be run in parallel.

- Time complexity of RNN: $\mathcal{O}(L \cdot$ time per step).
- Time complexity of SSM: $\mathcal{O}(L +$ time per step).

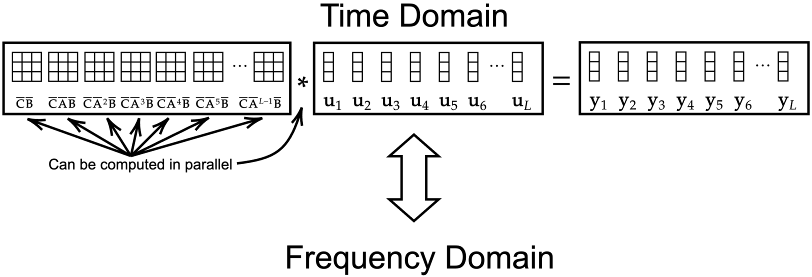
Efficiency of SSMs

An LTI system is linear. Hence, it can be evaluated more easily.



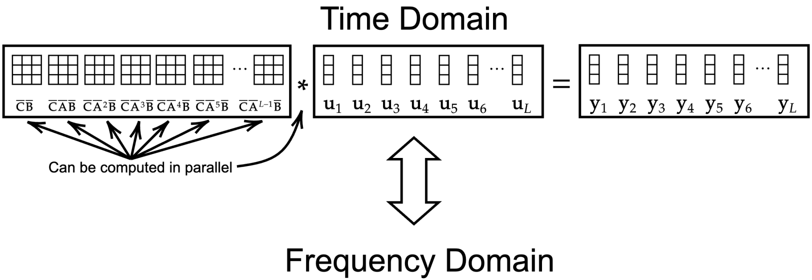
Efficiency of SSMs

An LTI system is linear. Hence, it can be evaluated more easily.



Efficiency of SSMs

An LTI system is linear. Hence, it can be evaluated more easily.



Stay here for the second half of the tutorial!

Training Stability of SSMs

Training Stability of SSMs

The gradient is the gradient. It doesn't matter how you compute it.

Training Stability of SSMs

The gradient is the gradient. It doesn't matter how you compute it. Then, why doesn't an SSM suffer from the vanishing and exploding gradient issues?

Training Stability of SSMs

The gradient is the gradient. It doesn't matter how you compute it. Then, why doesn't an SSM suffer from the vanishing and exploding gradient issues?

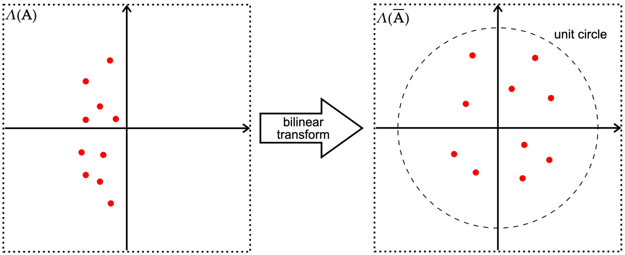
Answer: by discretizing the system with a small Δt!

Training Stability of SSMs

The gradient is the gradient. It doesn't matter how you compute it. Then, why doesn't an SSM suffer from the vanishing and exploding gradient issues?

Answer: by discretizing the system with a small Δt!

- By restricting $\Lambda(\mathbf{A})$ in the left half-plane, we guarantee that $\rho(\overline{\mathbf{A}}) < 1$.

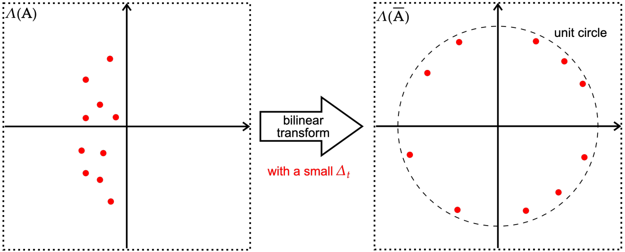


Training Stability of SSMs

The gradient is the gradient. It doesn't matter how you compute it. Then, why doesn't an SSM suffer from the vanishing and exploding gradient issues?

Answer: by discretizing the system with a small Δt!

- By restricting $\Lambda(\mathbf{A})$ in the left half-plane, we guarantee that $\rho(\overline{\mathbf{A}}) < 1$.
- By setting Δt small, we have that $\rho(\overline{\mathbf{A}})$ is close to one.



Training Stability of SSMs

The gradient is the gradient. It doesn't matter how you compute it. Then, why doesn't an SSM suffer from the vanishing and exploding gradient issues?

Answer: by discretizing the system with a small Δt!

- By restricting $\Lambda(\mathbf{A})$ in the left half-plane, we guarantee that $\rho(\overline{\mathbf{A}}) < 1$.
- By setting Δt small, we have that $\rho(\overline{\mathbf{A}})$ is close to one.

Training Stability of SSMs

The gradient is the gradient. It doesn't matter how you compute it. Then, why doesn't an SSM suffer from the vanishing and exploding gradient issues?

Answer: by discretizing the system with a small Δt!

- By restricting $\Lambda(\mathbf{A})$ in the left half-plane, we guarantee that $\rho(\overline{\mathbf{A}}) < 1$.
- By setting Δt small, we have that $\rho(\overline{\mathbf{A}})$ is close to one.

Stay here for the second half of the tutorial!

Mambas

Mambas

SSMs are good at learning tasks that involve long-range dependencies, but their vanilla forms do not lead to good language models.

Mambas

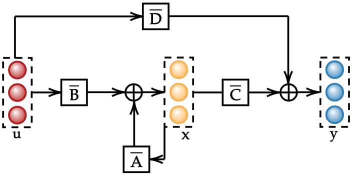
SSMs are good at learning tasks that involve long-range dependencies, but their vanilla forms do not lead to good language models.
One of the reasons is that in an SSM, every element in a sequence is processed using the same mechanism. The Mamba models [Gu and Dao, 2023] fix this issue by letting $\mathbf{B}$ and $\mathbf{C}$ depend on the input.

Mambas

SSMs are good at learning tasks that involve long-range dependencies, but their vanilla forms do not lead to good language models.
One of the reasons is that in an SSM, every element in a sequence is processed using the same mechanism. The Mamba models [Gu and Dao, 2023] fix this issue by letting **B** and **C** depend on the input.

SSM

$$\mathbf{x}'(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t)$$
$$\mathbf{y}(t) = \mathbf{C}\mathbf{x}(t) + \mathbf{D}\mathbf{u}(t)$$

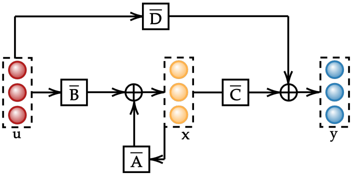


Mambas

SSMs are good at learning tasks that involve long-range dependencies, but their vanilla forms do not lead to good language models.
One of the reasons is that in an SSM, every element in a sequence is processed using the same mechanism. The Mamba models [Gu and Dao, 2023] fix this issue by letting **B** and **C** depend on the input.

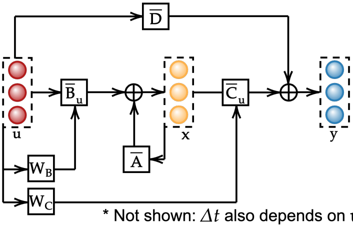
SSM

$$\mathbf{x}'(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t)$$
$$\mathbf{y}(t) = \mathbf{C}\mathbf{x}(t) + \mathbf{D}\mathbf{u}(t)$$



Mamba

$$\mathbf{x}'(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}(\mathbf{u}(t))\mathbf{u}(t)$$
$$\mathbf{y}(t) = \mathbf{C}(\mathbf{u}(t))\mathbf{x}(t) + \mathbf{D}\mathbf{u}(t)$$



* Not shown: Δt also depends on $\mathbf{u}$

Properties of Mamba

Properties of Mamba

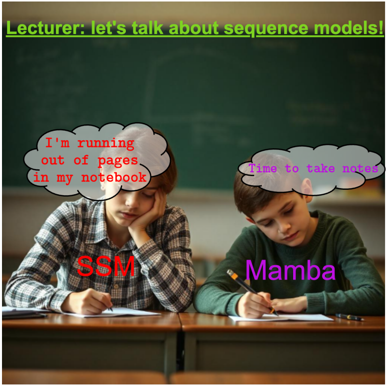
The dynamical system in Mamba is time-variant. Hence, it cannot be evaluated using a convolution. However, efficient parallel algorithms exist.

Properties of Mamba

The dynamical system in Mamba is time-variant. Hence, it cannot be evaluated using a convolution. However, efficient parallel algorithms exist. Right now, the success of Mamba is justified by its capability of "selectively memorizing" the sequence.

Properties of Mamba

The dynamical system in Mamba is time-variant. Hence, it cannot be evaluated using a convolution. However, efficient parallel algorithms exist. Right now, the success of Mamba is justified by its capability of "selectively memorizing" the sequence.



Outline of Part II

1. Recap of state-space models
2. The “real” story
3. The “imaginary” story

Recap of State-Space Models

Linear, Time-Invariant Systems

Linear, Time-Invariant Systems

A state space model (SSM) [Gu et al., 2022] leverages linear, time-variant (LTI) systems as its recurrent unit:

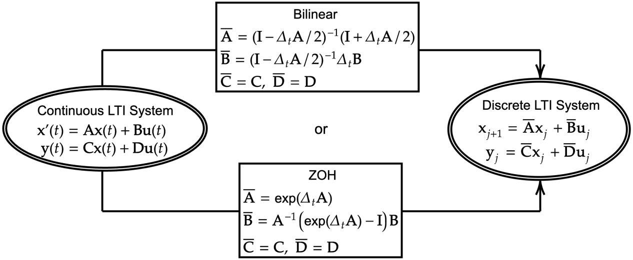
$$\begin{aligned}\mathbf{x}'(t) &= \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t),\\ \mathbf{y}(t) &= \mathbf{C}\mathbf{x}(t) + \mathbf{D}\mathbf{u}(t).\end{aligned}$$

Linear, Time-Invariant Systems

A state space model (SSM) [Gu et al., 2022] leverages linear, time-variant (LTI) systems as its recurrent unit:

$$\mathbf{x}'(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t),$$
$$\mathbf{y}(t) = \mathbf{C}\mathbf{x}(t) + \mathbf{D}\mathbf{u}(t).$$

We have to discretize the system with respect to some trainable sampling period $\Delta t > 0$:



SSMs vs RNNs

SSMs vs RNNs

RNN

$$\mathbf{x}_k = \tanh(\mathbf{W}_1 \mathbf{x}_{k-1} + \mathbf{U}\mathbf{u}_k + \mathbf{b}_1)$$
$$\mathbf{y}_k = \mathrm{ReLU}(\mathbf{W}_2 \mathbf{x}_k + \mathbf{b}_2)$$

SSMs vs RNNs

RNN

$$\mathbf{x}_k = \tanh(\mathbf{W}_1 \mathbf{x}_{k-1} + \mathbf{U}\mathbf{u}_k + \mathbf{b}_1)$$
$$\mathbf{y}_k = \mathrm{ReLU}(\mathbf{W}_2 \mathbf{x}_k + \mathbf{b}_2)$$

SSM

$$\mathbf{x}'(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t)$$
$$\mathbf{y}(t) = \mathbf{C}\mathbf{x}(t) + \mathbf{D}\mathbf{u}(t)$$

or

$$\mathbf{x}_k = \overline{\mathbf{A}}\mathbf{x}_{k-1} + \overline{\mathbf{B}}\mathbf{u}_k$$
$$\mathbf{y}_k = \overline{\mathbf{C}}\mathbf{x}_k + \overline{\mathbf{D}}\mathbf{u}_k$$

SSMs vs RNNs

RNN

$$\mathbf{x}_k = \tanh(\mathbf{W}_1 \mathbf{x}_{k-1} + \mathbf{U}\mathbf{u}_k + \mathbf{b}_1)$$
$$\mathbf{y}_k = \mathrm{ReLU}(\mathbf{W}_2 \mathbf{x}_k + \mathbf{b}_2)$$

SSM

$$\mathbf{x}'(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t)$$
$$\mathbf{y}(t) = \mathbf{C}\mathbf{x}(t) + \mathbf{D}\mathbf{u}(t)$$

or

$$\mathbf{x}_k = \overline{\mathbf{A}}\mathbf{x}_{k-1} + \overline{\mathbf{B}}\mathbf{u}_k$$
$$\mathbf{y}_k = \overline{\mathbf{C}}\mathbf{x}_k + \overline{\mathbf{D}}\mathbf{u}_k$$

SSMs vs RNNs

RNN

$$\mathbf{x}_k = \tanh(\mathbf{W}_1 \mathbf{x}_{k-1} + \mathbf{U}\mathbf{u}_k + \mathbf{b}_1)$$
$$\mathbf{y}_k = \mathrm{ReLU}(\mathbf{W}_2 \mathbf{x}_k + \mathbf{b}_2)$$

SSM

$$\mathbf{x}'(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t)$$
$$\mathbf{y}(t) = \mathbf{C}\mathbf{x}(t) + \mathbf{D}\mathbf{u}(t)$$

or

$$\mathbf{x}_k = \overline{\mathbf{A}}\mathbf{x}_{k-1} + \overline{\mathbf{B}}\mathbf{u}_k$$
$$\mathbf{y}_k = \overline{\mathbf{C}}\mathbf{x}_k + \overline{\mathbf{D}}\mathbf{u}_k$$

What are the main differences between an RNN and an SSM?

SSMs vs RNNs

RNN

$$\mathbf{x}_k = \tanh(\mathbf{W}_1 \mathbf{x}_{k-1} + \mathbf{U}\mathbf{u}_k + \mathbf{b}_1)$$
$$\mathbf{y}_k = \mathrm{ReLU}(\mathbf{W}_2 \mathbf{x}_k + \mathbf{b}_2)$$

SSM

$$\mathbf{x}'(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t)$$
$$\mathbf{y}(t) = \mathbf{C}\mathbf{x}(t) + \mathbf{D}\mathbf{u}(t)$$

or

$$\mathbf{x}_k = \overline{\mathbf{A}}\mathbf{x}_{k-1} + \overline{\mathbf{B}}\mathbf{u}_k$$
$$\mathbf{y}_k = \overline{\mathbf{C}}\mathbf{x}_k + \overline{\mathbf{D}}\mathbf{u}_k$$

What are the main differences between an RNN and an SSM?

1. An RNN is nonlinear while an SSM is linear.

SSMs vs RNNs

RNN	SSM
$\mathbf{x}_k = \tanh(\mathbf{W}_1\mathbf{x}_{k-1} + \mathbf{U}\mathbf{u}_k + \mathbf{b}_1)$ $\mathbf{y}_k = \mathrm{ReLU}(\mathbf{W}_2\mathbf{x}_k + \mathbf{b}_2)$	$\mathbf{x}'(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t)$ $\mathbf{y}(t) = \mathbf{C}\mathbf{x}(t) + \mathbf{D}\mathbf{u}(t)$ or $\mathbf{x}_k = \overline{\mathbf{A}}\mathbf{x}_{k-1} + \overline{\mathbf{B}}\mathbf{u}_k$ $\mathbf{y}_k = \overline{\mathbf{C}}\mathbf{x}_k + \overline{\mathbf{D}}\mathbf{u}_k$

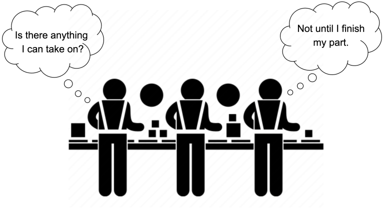
What are the main differences between an RNN and an SSM?

1. An RNN is nonlinear while an SSM is linear.
2. An RNN is completely discrete while an SSM has an underlying continuous system.

Two Weaknesses of RNNs

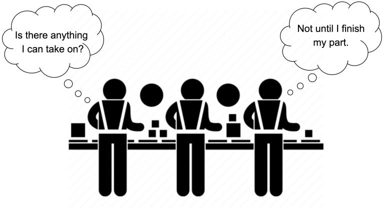
Two Weaknesses of RNNs

1. We have to backpropagate through an RNN recurrently. Assuming a sequence as a length of L, it takes $\mathcal{O}(L \cdot \text{time per step})$.



Two Weaknesses of RNNs

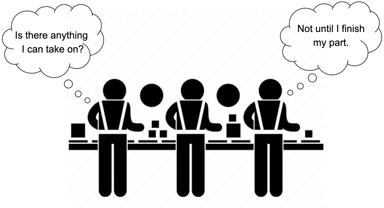
1. We have to backpropagate through an RNN recurrently. Assuming a sequence as a length of L, it takes $\mathcal{O}(L \cdot \text{time per step})$.



2. An RNN suffers from the exploding or vanishing gradient issues, impairing the training stability or the long-range memory retention.

Two Weaknesses of RNNs

1. We have to backpropagate through an RNN recurrently. Assuming a sequence as a length of L, it takes $\mathcal{O}(L \cdot$ time per step).

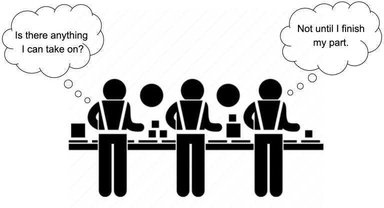


2. An RNN suffers from the exploding or vanishing gradient issues, impairing the training stability or the long-range memory retention.

If $\rho(\mathbf{W}) > 1$, then $\|\mathbf{W}^j\|_2$ explodes exponentially as $j \to \infty$.

Two Weaknesses of RNNs

1. We have to backpropagate through an RNN recurrently. Assuming a sequence as a length of L, it takes $\mathcal{O}(L \cdot \text{time per step})$.



2. An RNN suffers from the exploding or vanishing gradient issues, impairing the training stability or the long-range memory retention.

If $\rho(\mathbf{W}) > 1$, then $\|\mathbf{W}^j\|_2$ explodes exponentially as $j \to \infty$.

If $\rho(\mathbf{W}) < 1$, then $\|\mathbf{W}^j\|_2$ vanishes exponentially as $j \to \infty$.

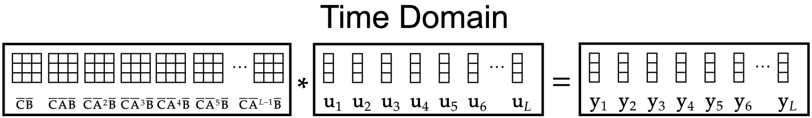
Efficiency of SSMs

Efficiency of SSMs

An LTI system is linear. Hence, it can be evaluated more easily.

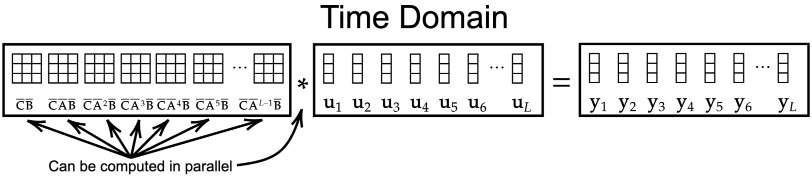
Efficiency of SSMs

An LTI system is linear. Hence, it can be evaluated more easily.



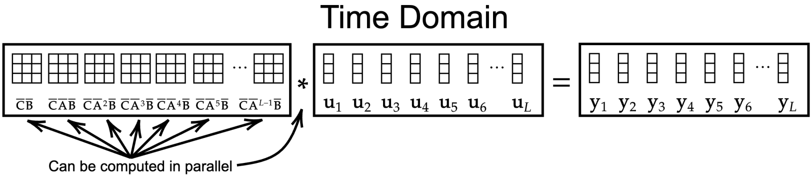
Efficiency of SSMs

An LTI system is linear. Hence, it can be evaluated more easily.



Efficiency of SSMs

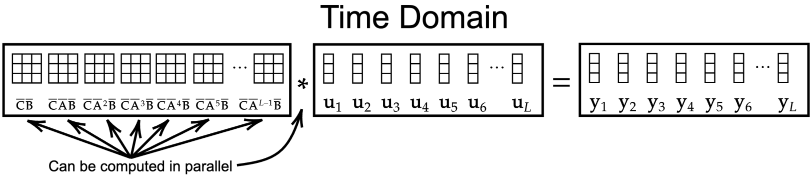
An LTI system is linear. Hence, it can be evaluated more easily.



Assume we have L processors that can be run in parallel.

Efficiency of SSMs

An LTI system is linear. Hence, it can be evaluated more easily.

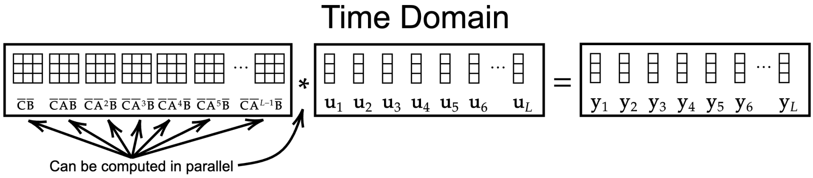


Assume we have L processors that can be run in parallel.

- Time complexity of RNN: $\mathcal{O}(L \cdot$ time per step).

Efficiency of SSMs

An LTI system is linear. Hence, it can be evaluated more easily.

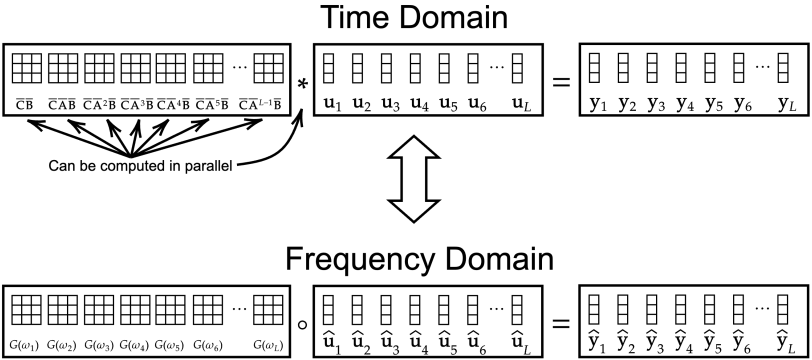


Assume we have L processors that can be run in parallel.

- Time complexity of RNN: $\mathcal{O}(L \cdot$ time per step).
- Time complexity of SSM: $\mathcal{O}(L +$ time per step).

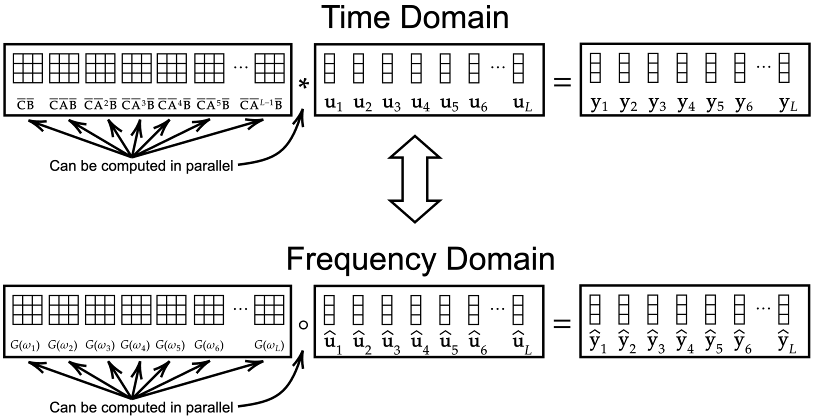
Efficiency of SSMs

An LTI system is linear. Hence, it can be evaluated more easily.



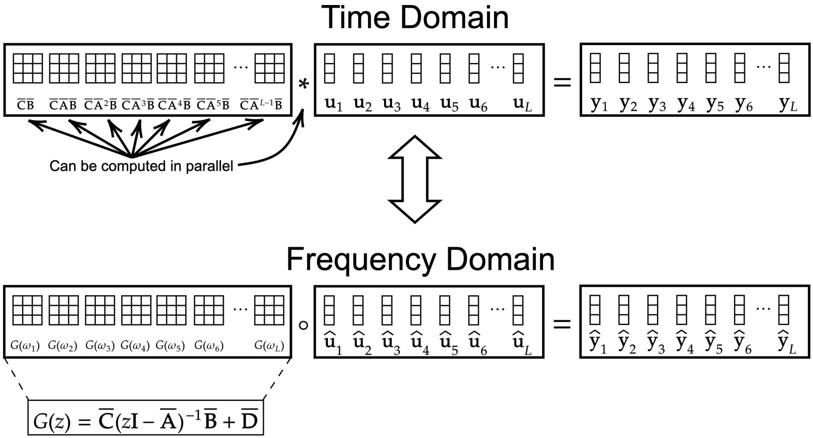
Efficiency of SSMs

An LTI system is linear. Hence, it can be evaluated more easily.



Efficiency of SSMs

An LTI system is linear. Hence, it can be evaluated more easily.



An SSM can be Made Deep

An SSM can be Made Deep

The LTI system is linear.

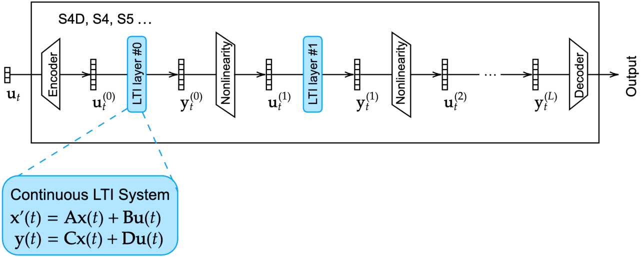
It can't capture nonlinear dynamics.

It is not as expressive as an RNN.

An SSM can be Made Deep

The LTI system is linear.
It can't capture nonlinear dynamics.
It is not as expressive as an RNN.

An LTI system is linear, but an SSM is not.



Training Stability of SSMs

Training Stability of SSMs

The gradient is the gradient. It doesn't matter how you compute it.

Training Stability of SSMs

The gradient is the gradient. It doesn't matter how you compute it. Then, why doesn't an SSM suffer from the vanishing and exploding gradient issues?

Training Stability of SSMs

The gradient is the gradient. It doesn't matter how you compute it. Then, why doesn't an SSM suffer from the vanishing and exploding gradient issues?

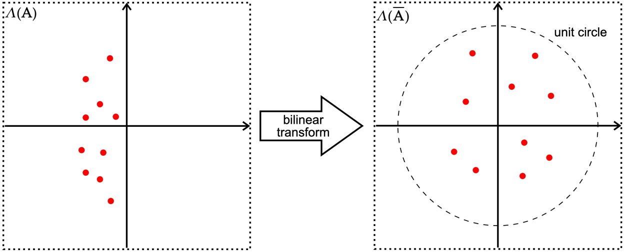
Answer: by discretizing the system with a small Δt!

Training Stability of SSMs

The gradient is the gradient. It doesn't matter how you compute it. Then, why doesn't an SSM suffer from the vanishing and exploding gradient issues?

Answer: by discretizing the system with a small Δt!

- By restricting $\Lambda(\mathbf{A})$ in the left half-plane, we guarantee that $\rho(\overline{\mathbf{A}}) < 1$.

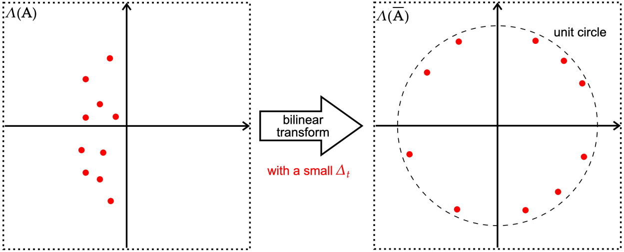


Training Stability of SSMs

The gradient is the gradient. It doesn't matter how you compute it. Then, why doesn't an SSM suffer from the vanishing and exploding gradient issues?

Answer: by discretizing the system with a small Δt!

- By restricting $\Lambda(\mathbf{A})$ in the left half-plane, we guarantee that $\rho(\overline{\mathbf{A}}) < 1$.
- By setting Δt small, we have that $\rho(\overline{\mathbf{A}})$ is close to one.



SSMs can Capture the Long-Range Dependency

SSMs can Capture the Long-Range Dependency

Long-Range Dependency $\neq$ Long-Range Sequence

SSMs can Capture the Long-Range Dependency

Long-Range Dependency $\neq$ Long-Range Sequence

Model (Input length)	ListOps (2,048)	Text (4,096)	Retrieval (4,000)	Image (1,024)	Pathfinder (1,024)	Path-X (16,384)	Avg.
Transformer	36.37	64.27	57.46	42.44	71.40	✗	53.66
Luna-256	37.25	64.57	79.29	47.38	77.72	✗	59.37
H-Trans.-1D	49.53	78.69	63.99	46.05	68.78	✗	61.41
CCNN	43.60	84.08	✗	88.90	91.51	✗	68.02
Mega ($\mathcal{O}(L^2)$)	**63.14**	**90.43**	<u>91.25</u>	**90.44**	**96.01**	<u>97.98</u>	**88.21**
Mega-chunk ($\mathcal{O}(L)$)	58.76	<u>90.19</u>	90.97	85.80	94.41	93.81	85.66
S4D-LegS	60.47	86.18	89.46	88.19	93.06	91.95	84.89
S4-LegS	59.60	86.82	90.90	88.65	94.20	96.35	86.09
Liquid-S4	<u>62.75</u>	89.02	91.20	<u>89.50</u>	94.8	96.66	87.32
S5	62.15	89.31	**91.40**	88.00	<u>95.33</u>	**98.58**	<u>87.46</u>

Overview of the Next Two Parts

Overview of the Next Two Parts

We saw that one can compute an LTI system from its transfer function:

$$\hat{\mathbf{y}}(s) = \mathbf{G}(is)\hat{\mathbf{u}}(s), \qquad \mathbf{G}(is) = \mathbf{C}(is\mathbf{I} - \mathbf{A})^{-1}\mathbf{B} + \mathbf{D}.$$

Overview of the Next Two Parts

We saw that one can compute an LTI system from its transfer function:

$$\hat{\mathbf{y}}(s) = \mathbf{G}(is)\hat{\mathbf{u}}(s), \qquad \mathbf{G}(is) = \mathbf{C}(is\mathbf{I} - \mathbf{A})^{-1}\mathbf{B} + \mathbf{D}.$$

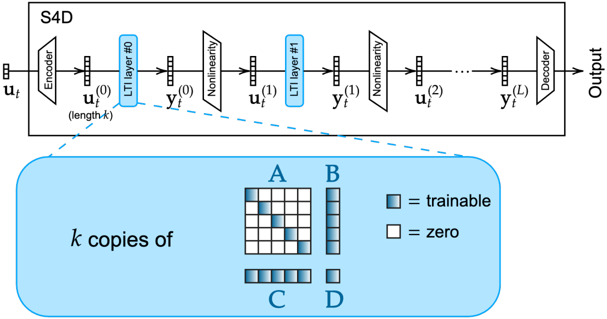
A key question is: how can we efficiently sample $\mathbf{G}$?

Overview of the Next Two Parts

We saw that one can compute an LTI system from its transfer function:

$$\hat{\mathbf{y}}(s) = \mathbf{G}(is)\hat{\mathbf{u}}(s), \qquad \mathbf{G}(is) = \mathbf{C}(is\mathbf{I} - \mathbf{A})^{-1}\mathbf{B} + \mathbf{D}.$$

A key question is: how can we efficiently sample $\mathbf{G}$?
From now on, we assume that an LTI system is single-input/single-output (SISO). Moreover, the matrix $\mathbf{A}$ is diagonal.

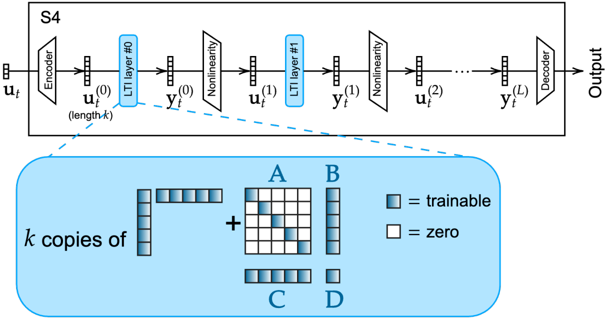


Overview of the Next Two Parts

We saw that one can compute an LTI system from its transfer function:

$$\hat{\mathbf{y}}(s) = \mathbf{G}(is)\hat{\mathbf{u}}(s), \qquad \mathbf{G}(is) = \mathbf{C}(is\mathbf{I} - \mathbf{A})^{-1}\mathbf{B} + \mathbf{D}.$$

A key question is: how can we efficiently sample $\mathbf{G}$?
From now on, we assume that an LTI system is single-input/single-output (SISO). Moreover, the matrix $\mathbf{A}$ is diagonal.

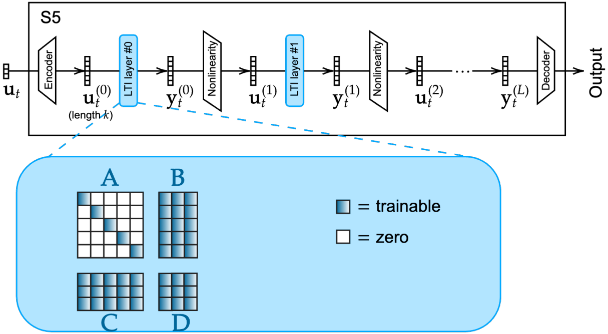


Overview of the Next Two Parts

We saw that one can compute an LTI system from its transfer function:

$$\hat{\mathbf{y}}(s) = \mathbf{G}(is)\hat{\mathbf{u}}(s), \qquad \mathbf{G}(is) = \mathbf{C}(is\mathbf{I} - \mathbf{A})^{-1}\mathbf{B} + \mathbf{D}.$$

A key question is: how can we efficiently sample $\mathbf{G}$?
From now on, we assume that an LTI system is single-input/single-output (SISO). Moreover, the matrix $\mathbf{A}$ is diagonal.



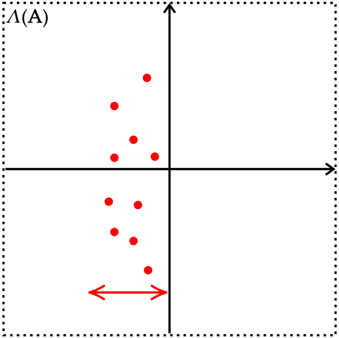
Overview of the Next Two Parts

Overview of the Next Two Parts

As mentioned earlier, some key insights could be obtained by studying the spectrum of $\mathbf{A}$. When $\mathbf{A} = \text{diag}(a_1, \ldots, a_n)$ is diagonal, we have $\Lambda(\mathbf{A}) = \{a_1, \ldots, a_n\}$.

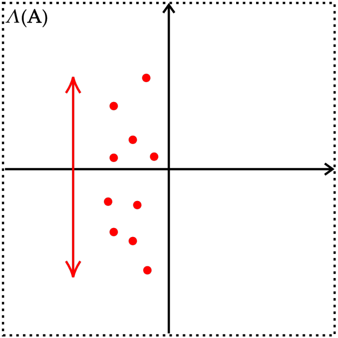
Overview of the Next Two Parts

As mentioned earlier, some key insights could be obtained by studying the spectrum of $\mathbf{A}$. When $\mathbf{A} = \text{diag}(a_1, \ldots, a_n)$ is diagonal, we have $\Lambda(\mathbf{A}) = \{a_1, \ldots, a_n\}$.



Part II

The "Real" Story



Part III

The "Imaginary" Story

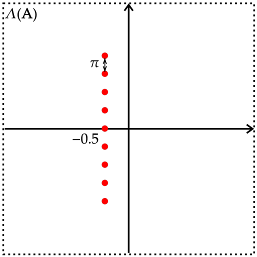
The "Real" Story

cf. *HOPE for a Robust Parameterization of Long-memory State Space Models*

Initializing an SSM

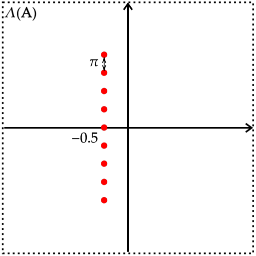
Initializing an SSM

SSMs are very sensitive to initialization. You may have heard of the so-called HiPPO initialization.



Initializing an SSM

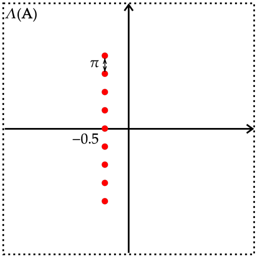
SSMs are very sensitive to initialization. You may have heard of the so-called HiPPO initialization.



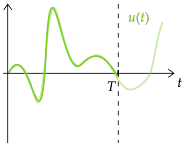
Traditionally, HiPPO was justified by the idea of "projecting onto orthogonal polynomials and storing the polynomial coefficients."

Initializing an SSM

SSMs are very sensitive to initialization. You may have heard of the so-called HiPPO initialization.

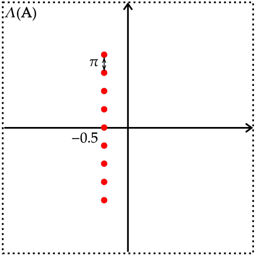


Traditionally, HiPPO was justified by the idea of "projecting onto orthogonal polynomials and storing the polynomial coefficients."

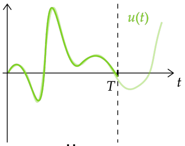


Initializing an SSM

SSMs are very sensitive to initialization. You may have heard of the so-called HiPPO initialization.



Traditionally, HiPPO was justified by the idea of "projecting onto orthogonal polynomials and storing the polynomial coefficients."

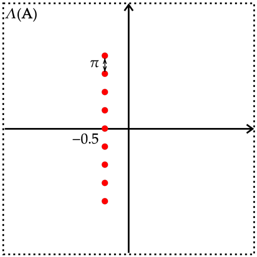


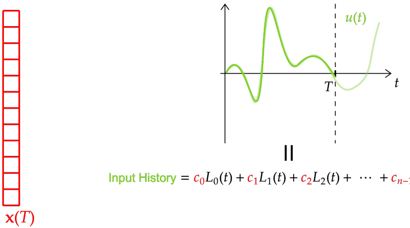
$$\text{Input History} = c_0 L_0(t) + c_1 L_1(t) + c_2 L_2(t) + \cdots + c_{n-1} L_{n-1}(t) + \cdots$$

Initializing an SSM

SSMs are very sensitive to initialization. You may have heard of the so-called HiPPO initialization.

Traditionally, HiPPO was justified by the idea of "projecting onto orthogonal polynomials and storing the polynomial coefficients."

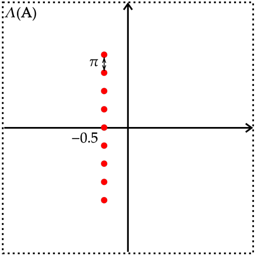




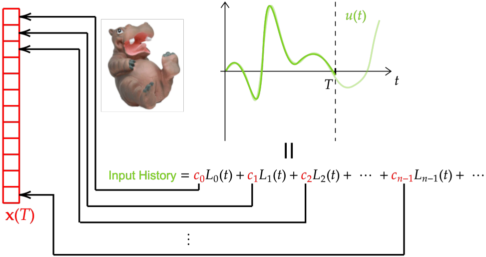
$$\text{Input History} = c_0 L_0(t) + c_1 L_1(t) + c_2 L_2(t) + \cdots + c_{n-1} L_{n-1}(t) + \cdots$$

Initializing an SSM

SSMs are very sensitive to initialization. You may have heard of the so-called HiPPO initialization.

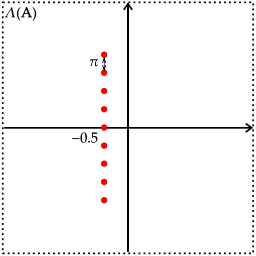


Traditionally, HiPPO was justified by the idea of "projecting onto orthogonal polynomials and storing the polynomial coefficients."

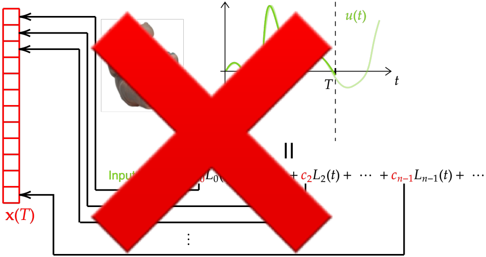


Initializing an SSM

SSMs are very sensitive to initialization. You may have heard of the so-called HiPPO initialization.



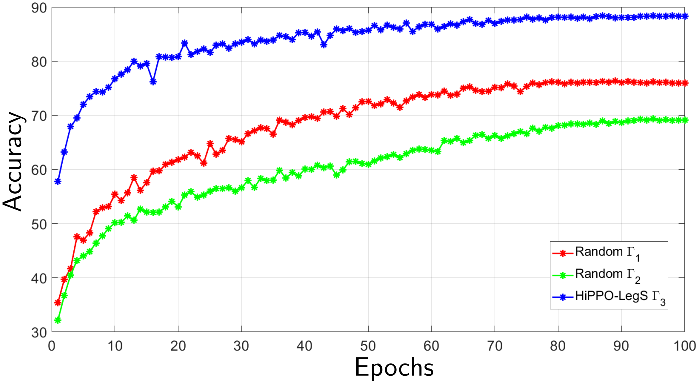
Traditionally, HiPPO was justified by the idea of "projecting onto orthogonal polynomials and storing the polynomial coefficients."



A Mystery

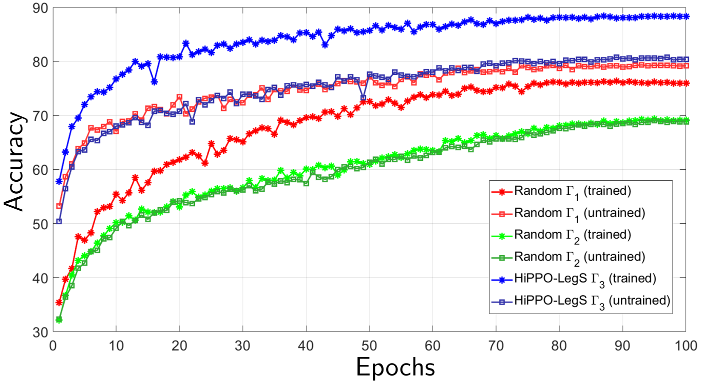
A Mystery

We train an SSM to learn the sequential CIFAR-10 task. We use different LTI systems at initialization.



A Mystery

We train an SSM to learn the sequential CIFAR-10 task. We use different LTI systems at initialization.



Hankel Singular Values

Hankel Singular Values

- The Hankel operator associated with a continuous-time LTI system is

$$\mathbf{H} : L^2(0,\infty) \to L^2(0,\infty), \quad (\mathbf{Hv})(t) = \int_0^\infty \mathbf{C} \exp((t+\tau)\mathbf{A})\mathbf{Bv}(\tau)d\tau.$$

Hankel Singular Values

- The Hankel operator associated with a continuous-time LTI system is

$$\mathbf{H} : L^2(0,\infty) \to L^2(0,\infty), \quad (\mathbf{Hv})(t) = \int_0^\infty \mathbf{C}\exp((t+\tau)\mathbf{A})\mathbf{Bv}(\tau)d\tau.$$

- The Hankel matrix associated with a discrete LTI system is

$$\overline{\mathbf{H}} : \ell^2 \to \ell^2, \quad \overline{\mathbf{H}}_{i,j} = \overline{\mathbf{C}}\overline{\mathbf{A}}^{i+j}\overline{\mathbf{B}}, \qquad i,j \geq 0.$$

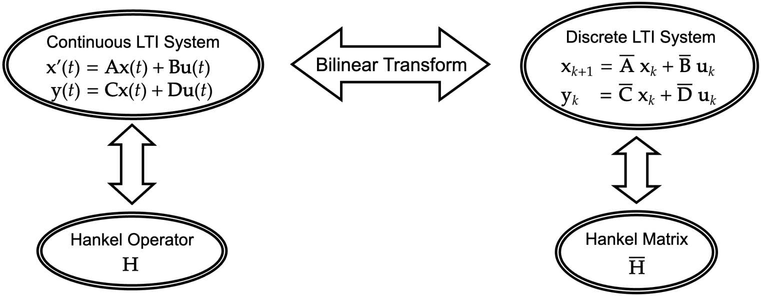
Hankel Singular Values

- The Hankel operator associated with a continuous-time LTI system is

$$\mathbf{H} : L^2(0,\infty) \to L^2(0,\infty), \quad (\mathbf{H}\mathbf{v})(t) = \int_0^\infty \mathbf{C}\exp((t+\tau)\mathbf{A})\mathbf{B}\mathbf{v}(\tau)d\tau.$$

- The Hankel matrix associated with a discrete LTI system is

$$\overline{\mathbf{H}} : \ell^2 \to \ell^2, \quad \overline{\mathbf{H}}_{i,j} = \overline{\mathbf{C}}\overline{\mathbf{A}}^{i+j}\overline{\mathbf{B}}, \qquad i,j \geq 0.$$



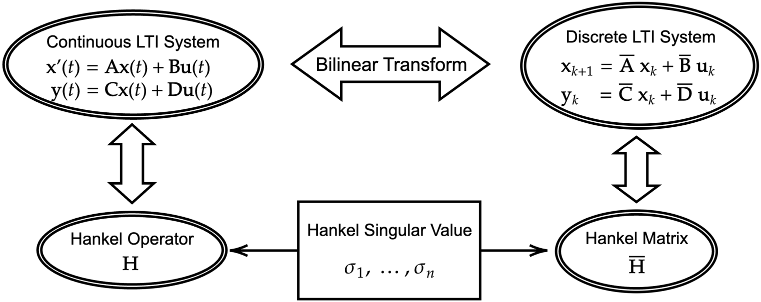
Hankel Singular Values

- The Hankel operator associated with a continuous-time LTI system is

$$\mathbf{H} : L^2(0,\infty) \to L^2(0,\infty), \quad (\mathbf{H}\mathbf{v})(t) = \int_0^\infty \mathbf{C}\exp((t+\tau)\mathbf{A})\mathbf{B}\mathbf{v}(\tau)d\tau.$$

- The Hankel matrix associated with a discrete LTI system is

$$\overline{\mathbf{H}} : \ell^2 \to \ell^2, \quad \overline{\mathbf{H}}_{i,j} = \overline{\mathbf{C}}\overline{\mathbf{A}}^{i+j}\overline{\mathbf{B}}, \qquad i,j \geq 0.$$



Reduced-Order Modeling with Hankel Singular Values

Reduced-Order Modeling with Hankel Singular Values

For any $k < n$, there exists an LTI system $\tilde{\Gamma} = (\tilde{\mathbf{A}}, \tilde{\mathbf{B}}, \tilde{\mathbf{C}}, \tilde{\mathbf{D}})$ with $\tilde{\mathbf{A}} \in \mathbb{C}^{k\times k}$, such that

$$\|G - \tilde{G}\|_\infty \leq \sum_{j=k+1}^{n} \sigma_j(\mathbf{H}) \leq (n-k)\sigma_{k+1}(\mathbf{H}),$$

where G and $\tilde{G}$ are the transfer functions of Γ and $\tilde{\Gamma}$, respectively, and $\|\cdot\|_\infty$ is the infinity norm over the imaginal axis.

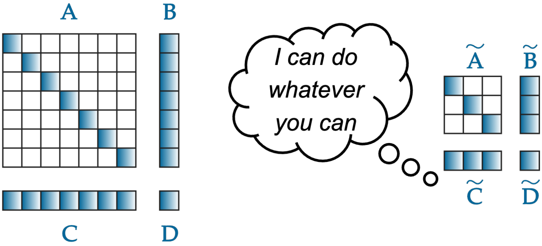
Reduced-Order Modeling with Hankel Singular Values

For any $k < n$, there exists an LTI system $\tilde{\Gamma} = (\tilde{\mathbf{A}}, \tilde{\mathbf{B}}, \tilde{\mathbf{C}}, \tilde{\mathbf{D}})$ with $\tilde{\mathbf{A}} \in \mathbb{C}^{k \times k}$, such that

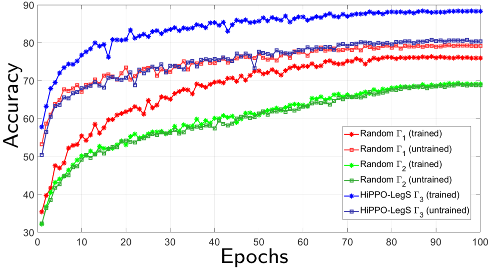
$$\|G - \tilde{G}\|_\infty \leq \sum_{j=k+1}^{n} \sigma_j(\mathbf{H}) \leq (n-k)\sigma_{k+1}(\mathbf{H}),$$

where G and $\tilde{G}$ are the transfer functions of Γ and $\tilde{\Gamma}$, respectively, and $\|\cdot\|_\infty$ is the infinity norm over the imaginal axis.

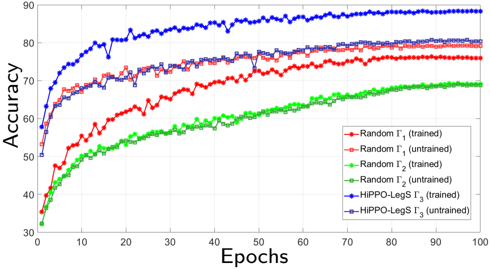
Hence, fast decaying Hankel singular values $\Rightarrow$ many states in $\mathbf{x}$ are redundant.



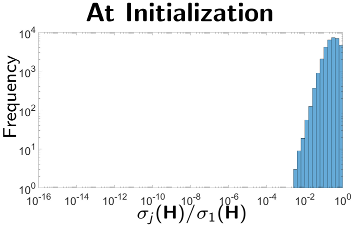
Unravel the Mystery

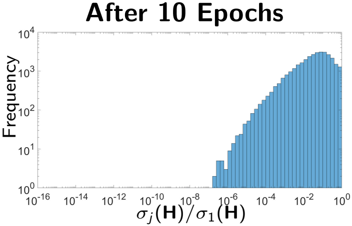


Unravel the Mystery

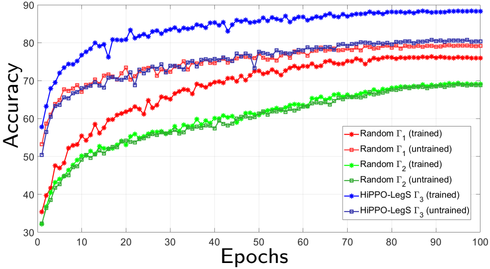


Hankel singular values of Γ_3:



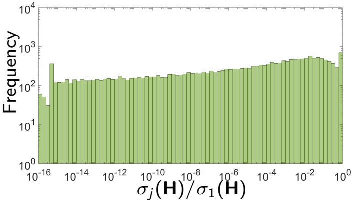


Unravel the Mystery

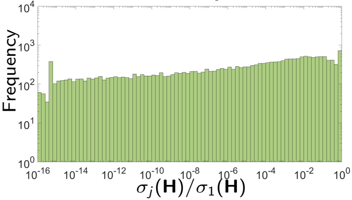


Hankel singular values of Γ_2:

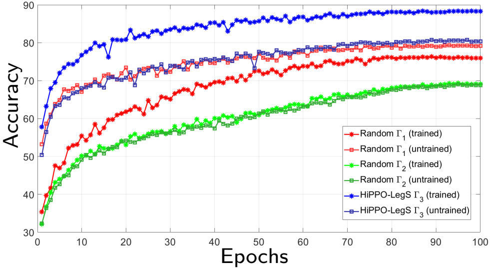
At Initialization



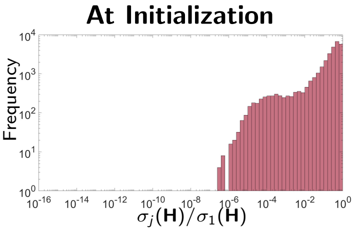
After 10 Epochs

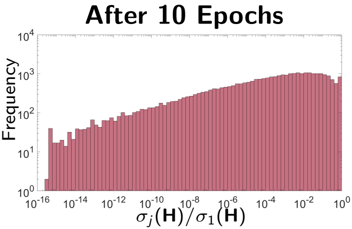


Unravel the Mystery



Hankel singular values of Γ_1:





Two Weaknesses of SSMs

Two Weaknesses of SSMs

1. From a random matrix theory perspective, high-rank LTI systems are scarce. Hence, even with a proper initialization, one can easily lose numerical ranks during training.

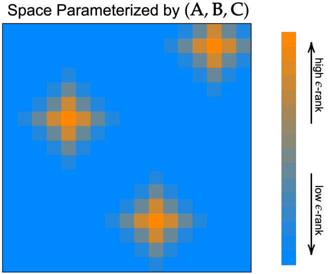
Two Weaknesses of SSMs

1. From a random matrix theory perspective, high-rank LTI systems are scarce. Hence, even with a proper initialization, one can easily lose numerical ranks during training.

The ϵ-rank of a random LTI system, i.e., the number of Hankel singular values σ_j with

$$\frac{\sigma_j}{\sigma_1} > \epsilon,$$

is roughly $\mathcal{O}(n^{1/2+\text{a bit}})$ with high probability.



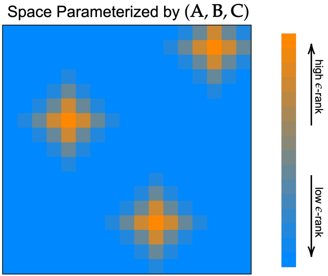
Two Weaknesses of SSMs

1. From a random matrix theory perspective, high-rank LTI systems are scarce. Hence, even with a proper initialization, one can easily lose numerical ranks during training.

The ϵ-rank of a random LTI system, i.e., the number of Hankel singular values σ_j with

$$\frac{\sigma_j}{\sigma_1} > \epsilon,$$

is roughly $\mathcal{O}(n^{1/2+\text{a bit}})$ with high probability.



Two Weaknesses of SSMs

2. The numerical stability of an LTI system depends on its parameters, making an SSM potentially not numerically stable over training.

Two Weaknesses of SSMs

2. The numerical stability of an LTI system depends on its parameters, making an SSM potentially not numerically stable over training.

When an LTI system in perturbed with

$$\|\mathbf{A} - \tilde{\mathbf{A}}\|_{\max} \leq \Delta_A, \qquad \|\mathbf{B} \circ \mathbf{C}^\top - \tilde{\mathbf{B}} \circ \tilde{\mathbf{C}}^\top\|_{\max} \leq \Delta_B.$$

The transfer function perturbation can be bounded by

$$\|G - \tilde{G}\|_\infty \leq n\Delta_B \max_j \frac{1}{|\mathrm{Re}(a_j)|} + 4n\Delta_A \max_j \frac{|b_j c_j|}{|\mathrm{Re}(a_j)|^2}.$$

Moreover, this bound is tight up to a factor of n.

Two Weaknesses of SSMs

2. The numerical stability of an LTI system depends on its parameters, making an SSM potentially not numerically stable over training.

> When an LTI system in perturbed with
>
> $$\|\mathbf{A} - \tilde{\mathbf{A}}\|_{\max} \leq \Delta_A, \qquad \|\mathbf{B} \circ \mathbf{C}^\top - \tilde{\mathbf{B}} \circ \tilde{\mathbf{C}}^\top\|_{\max} \leq \Delta_B.$$
>
> The transfer function perturbation can be bounded by
>
> $$\|G - \tilde{G}\|_\infty \leq n\Delta_B \max_j \frac{1}{|\mathrm{Re}(a_j)|} + 4n\Delta_A \max_j \frac{|b_j c_j|}{|\mathrm{Re}(a_j)|^2}.$$
>
> Moreover, this bound is tight up to a factor of n.

"[the Hankel singular values] decay more rapidly the farther the $\Lambda(\mathbf{A})$ falls in the left half of the complex plane." — [Baker et al., 2015]

HOPE State-Space Models

HOPE State-Space Models

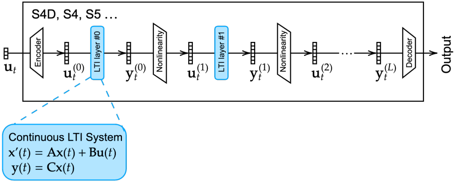
- Motivation: most LTI systems have low ranks and their numerical stability highly depends on the location of the poles a_j. Can we come up with a model that overcomes these issues?

HOPE State-Space Models

- Motivation: most LTI systems have low ranks and their numerical stability highly depends on the location of the poles a_j. Can we come up with a model that overcomes these issues?
- Solution: instead of parameterizing the LTI system using $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$, use a vector $\mathbf{h} \in \mathbb{C}^n$ to parameterize its Hankel matrix.

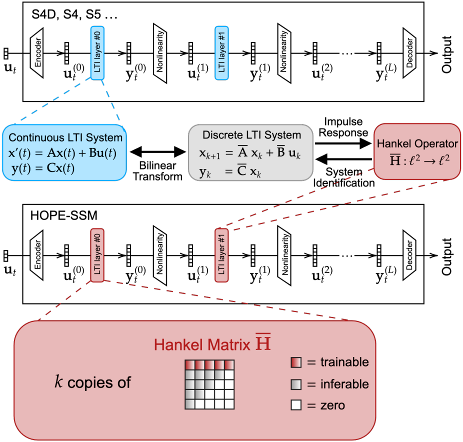
HOPE State-Space Models

- Motivation: most LTI systems have low ranks and their numerical stability highly depends on the location of the poles a_j. Can we come up with a model that overcomes these issues?
- Solution: instead of parameterizing the LTI system using $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$, use a vector $\mathbf{h} \in \mathbb{C}^n$ to parameterize its Hankel matrix.



HOPE State-Space Models

- Motivation: most LTI systems have low ranks and their numerical stability highly depends on the location of the poles a_j. Can we come up with a model that overcomes these issues?
- Solution: instead of parameterizing the LTI system using $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$, use a vector $\mathbf{h} \in \mathbb{C}^n$ to parameterize its Hankel matrix.

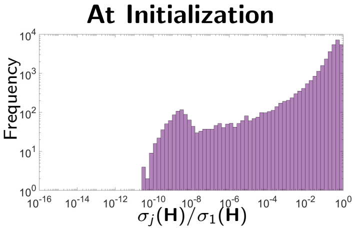


The Hopes of HOPE

The Hopes of HOPE

1. A Hankel matrix has slowly decaying singular values:

   The ϵ-rank of an $n \times n$ random Hankel matrix is almost surely $\Theta(n)$ as $n \to \infty$.



The Hopes of HOPE

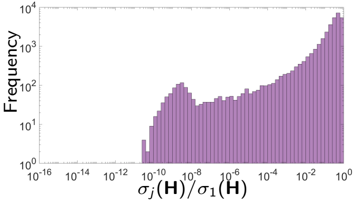
1. A Hankel matrix has slowly decaying singular values:

> The ϵ-rank of an $n \times n$ random Hankel matrix is almost surely $\Theta(n)$ as $n \to \infty$.

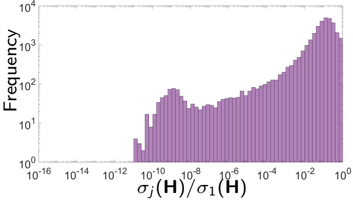
2. A Hankel matrix is perfectly stable to perturbation:

> Suppose we perturb $\mathbf{h}$ to $\tilde{\mathbf{h}}$. Then, we have
> $$\|G - \tilde{G}\|_\infty \leq \sqrt{n}\|\mathbf{h} - \tilde{\mathbf{h}}\|_2.$$

At Initialization

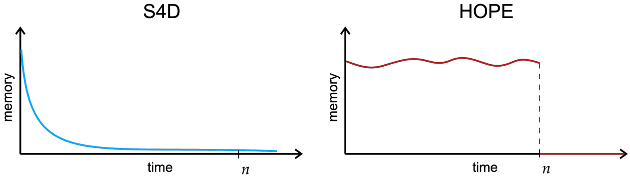


After 10 Epochs



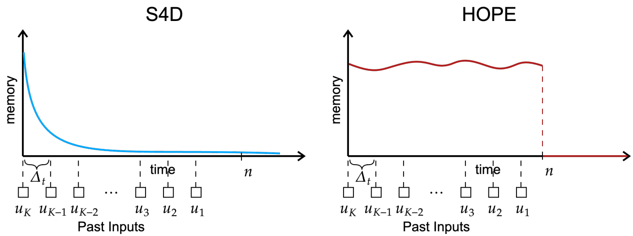
The Hopes of HOPE

3. A HOPE-SSM has slow-decaying memory.



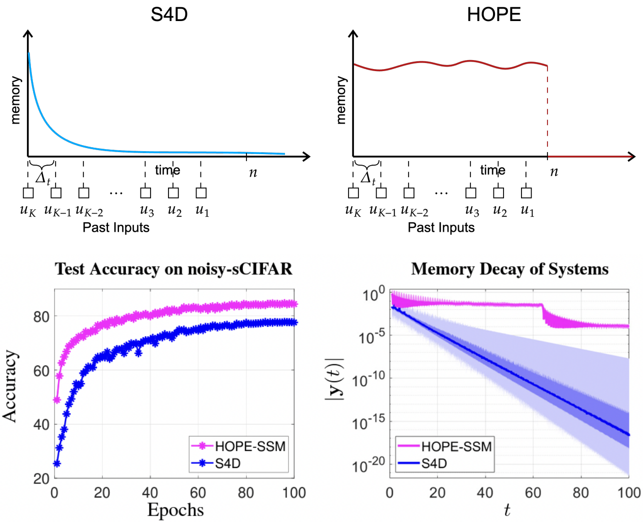
The Hopes of HOPE

3. A HOPE-SSM has slow-decaying memory.



The Hopes of HOPE

3. A HOPE-SSM has slow-decaying memory.



Another Interpretation of HOPE

Another Interpretation of HOPE

Recall that the transfer function $\overline{\mathbf{G}}(z)$ is a rational function. Different ways to parameterize an LTI system correspond to different ways to represent a rational function.

Another Interpretation of HOPE

Recall that the transfer function $\overline{\mathbf{G}}(z)$ is a rational function. Different ways to parameterize an LTI system correspond to different ways to represent a rational function.

Name	Formula	Parameterization	Models
Partial Fraction	$\sum_{j=1}^{n} \frac{b_j c_j}{z-a_j}$	diagonal $\mathbf{A}$	S4D/S5
Barycentric Formula	$\frac{\sum_{j=1}^{n} \frac{a_j}{z-z_j}}{1+\sum_{j=1}^{n} \frac{b_j}{z-z_j}}$	diag.-plus-rank-one $\mathbf{A}$	S4
Laurent Series	$\sum_{j=1}^{n} h_j z^{-j}$	Hankel matrix	HOPE

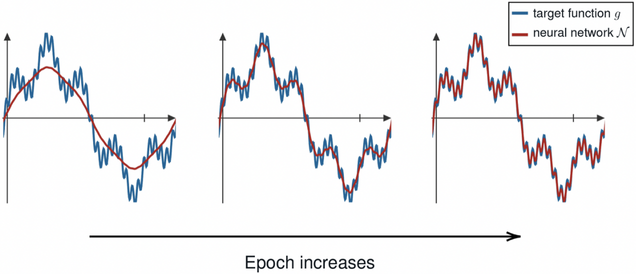
The "Imaginary" Story

cf. *Tuning Frequency Bias of State Space Models*

Why Do Neural Networks Generalize That Well?

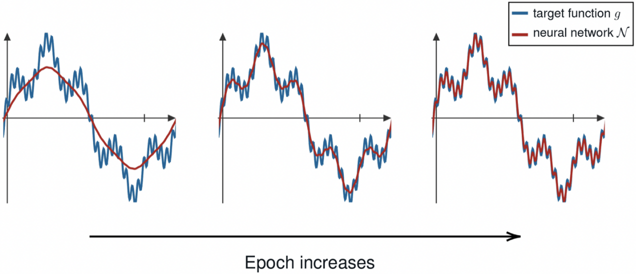
Why Do Neural Networks Generalize That Well?

One partial answer to the question from the title is called frequency bias:



Why Do Neural Networks Generalize That Well?

One partial answer to the question from the title is called frequency bias:

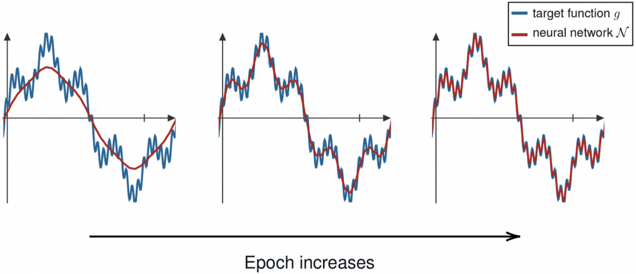


Good News.

Frequency bias prevents a NN from easily fitting high-frequency noises, making it good at generalization.

Why Do Neural Networks Generalize That Well?

One partial answer to the question from the title is called frequency bias:



Good News.

Frequency bias prevents a NN from easily fitting high-frequency noises, making it good at generalization.

Bad News.

Frequency bias puts a curse on learning useful high-frequency information in the target.

Do SSMs Have Frequency Bias?

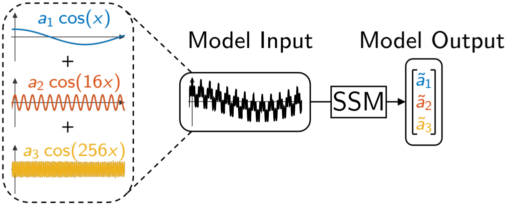
Do SSMs Have Frequency Bias?

There is a very natural notion of frequency for SSMs, i.e., the frequency along the time axis.

Do SSMs Have Frequency Bias?

There is a very natural notion of frequency for SSMs, i.e., the frequency along the time axis.
We observe that SSMs also have frequency bias.

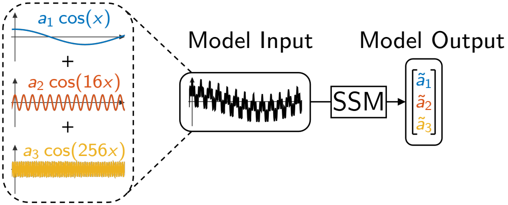
Problem Formulation



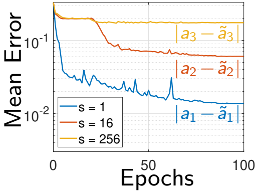
Do SSMs Have Frequency Bias?

There is a very natural notion of frequency for SSMs, i.e., the frequency along the time axis.
We observe that SSMs also have frequency bias.

Problem Formulation



Results



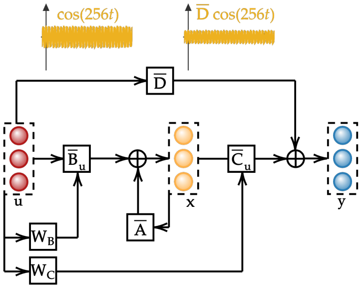
What is the Frequency Bias of an SSM?

What is the Frequency Bias of an SSM?

You may have imagined that frequency bias means that the output $\mathbf{y}(t)$ is of low frequency when the input $\mathbf{u}(t)$ contains high frequencies.

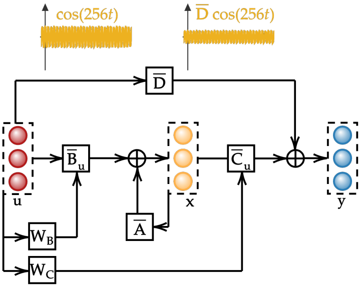
What is the Frequency Bias of an SSM?

You may have imagined that frequency bias means that the output $\mathbf{y}(t)$ is of low frequency when the input $\mathbf{u}(t)$ contains high frequencies.
Unfortunately, this is not the case.

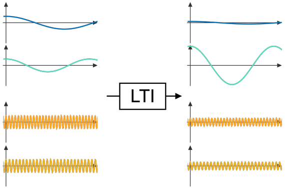


What is the Frequency Bias of an SSM?

You may have imagined that frequency bias means that the output $\mathbf{y}(t)$ is of low frequency when the input $\mathbf{u}(t)$ contains high frequencies.
Unfortunately, this is not the case.

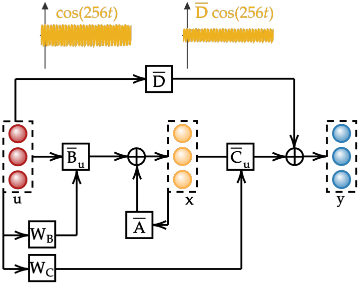


Frequency bias means an LTI system is better at distinguishing the low-frequency signals than the high-frequency ones.

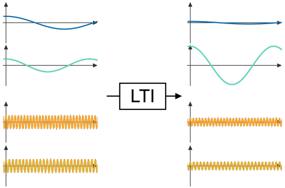


What is the Frequency Bias of an SSM?

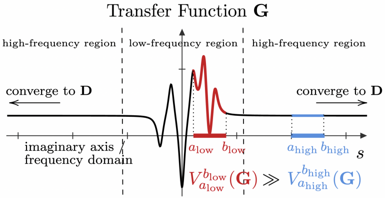
You may have imagined that frequency bias means that the output $\mathbf{y}(t)$ is of low frequency when the input $\mathbf{u}(t)$ contains high frequencies.
Unfortunately, this is not the case.



Frequency bias means an LTI system is better at distinguishing the low-frequency signals than the high-frequency ones.



Recall that $\hat{\mathbf{y}}(s) = \mathbf{G}(is)\hat{\mathbf{x}}(s)$.



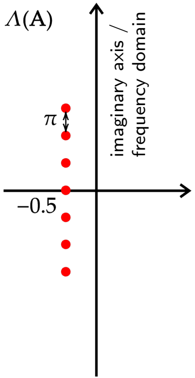
Why Do SSMs Have Frequency Bias?

Why Do SSMs Have Frequency Bias?

- An SSM is initialized with frequency bias.

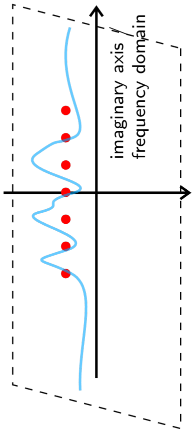
Why Do SSMs Have Frequency Bias?

- An SSM is initialized with frequency bias.



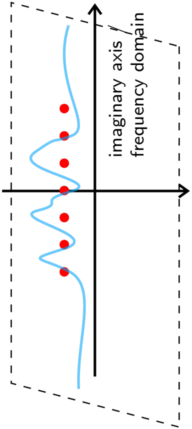
Why Do SSMs Have Frequency Bias?

- An SSM is initialized with frequency bias.



Why Do SSMs Have Frequency Bias?

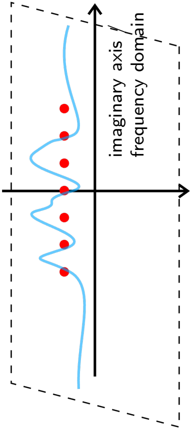
- An SSM is initialized with frequency bias.



- Will training push the eigenvalues of **A** to the high-frequency region?

Why Do SSMs Have Frequency Bias?

- An SSM is initialized with frequency bias.



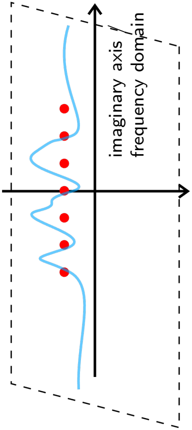
- Will training push the eigenvalues of **A** to the high-frequency region?

The gradient of a generic loss $\mathcal{L}$ with respect to $\mathrm{Im}(a_j)$ satisfies

$$\frac{\partial \mathcal{L}}{\partial \mathrm{Im}(a_j)} = \int_{-\infty}^{\infty} \frac{\partial \mathcal{L}}{\partial \mathbf{G}(is)} \cdot K_j(s)\, ds,$$
$$|K_j(s)| = \mathcal{O}\left(|s - \mathrm{Im}(a_j)|^{-2}\right).$$

Why Do SSMs Have Frequency Bias?

- An SSM is initialized with frequency bias.

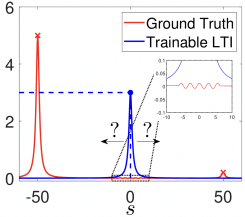


- Will training push the eigenvalues of **A** to the high-frequency region?

The gradient of a generic loss $\mathcal{L}$ with respect to $\mathrm{Im}(a_j)$ satisfies

$$\frac{\partial \mathcal{L}}{\partial \mathrm{Im}(a_j)} = \int_{-\infty}^{\infty} \frac{\partial \mathcal{L}}{\partial \mathbf{G}(is)} \cdot K_j(s)\, ds,$$
$$|K_j(s)| = \mathcal{O}\left(|s - \mathrm{Im}(a_j)|^{-2}\right).$$

Hence, a_j only learns "local" frequencies.



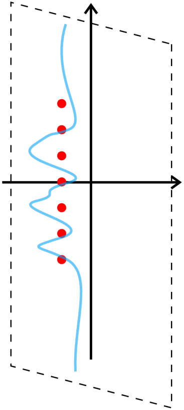
Tuning Frequency Bias via Initialization

Tuning Frequency Bias via Initialization

We can tune the frequency bias by scaling the initialization. In particular, we multiply each $\mathrm{Im}(a_j)$ by a hyperparameter $\alpha > 0$.

Tuning Frequency Bias via Initialization

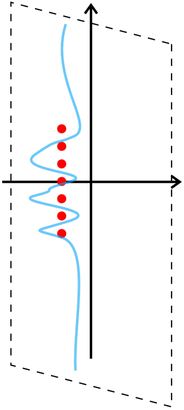
We can tune the frequency bias by scaling the initialization. In particular, we multiply each $\mathrm{Im}(a_j)$ by a hyperparameter $\alpha > 0$.



Default Bias

Tuning Frequency Bias via Initialization

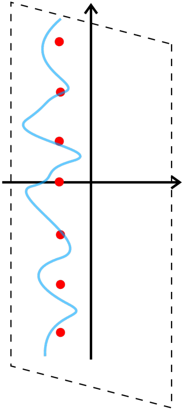
We can tune the frequency bias by scaling the initialization. In particular, we multiply each $\mathrm{Im}(a_j)$ by a hyperparameter $\alpha > 0$.



More Bias

Tuning Frequency Bias via Initialization

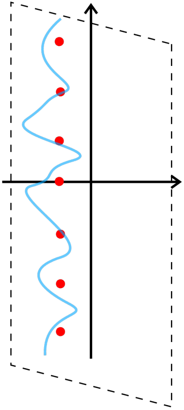
We can tune the frequency bias by scaling the initialization. In particular, we multiply each $\mathrm{Im}(a_j)$ by a hyperparameter $\alpha > 0$.



Less Bias

Tuning Frequency Bias via Initialization

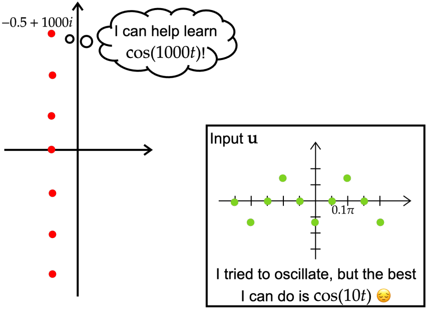
We can tune the frequency bias by scaling the initialization. In particular, we multiply each $\mathrm{Im}(a_j)$ by a hyperparameter $\alpha > 0$.



Less Bias

A Caveat

The eigenvalues of $\mathbf{A}$ should not be scaled arbitrarily large. In particular, they should not go beyond the Nyquist frequency.



Tuning Frequency Bias via Training

Tuning Frequency Bias via Training

We can apply a Sobolev-norm-based filter to the transfer function:

$$\hat{\mathbf{y}}(s) = (1 + |s|)^{\beta} \mathbf{G}(is)\hat{\mathbf{u}}(s).$$

Tuning Frequency Bias via Training

We can apply a Sobolev-norm-based filter to the transfer function:

$$\hat{\mathbf{y}}(s) = (1 + |s|)^{\beta} \mathbf{G}(is)\hat{\mathbf{u}}(s).$$

Intuitively, β reweighs the frequency domain.

Tuning Frequency Bias via Training

We can apply a Sobolev-norm-based filter to the transfer function:

$$\hat{\mathbf{y}}(s) = (1 + |s|)^{\beta} \mathbf{G}(is) \hat{\mathbf{u}}(s).$$

Intuitively, β reweighs the frequency domain.

- $\beta < 0 \Rightarrow$ low frequencies are weighted more, frequency bias is enhanced.

Tuning Frequency Bias via Training

We can apply a Sobolev-norm-based filter to the transfer function:

$$\hat{\mathbf{y}}(s) = (1 + |s|)^{\beta} \mathbf{G}(is)\hat{\mathbf{u}}(s).$$

Intuitively, β reweighs the frequency domain.

- $\beta < 0 \Rightarrow$ low frequencies are weighted more, frequency bias is enhanced.
- $\beta > 0 \Rightarrow$ low frequencies are weighted less, frequency bias is diminished.

Tuning Frequency Bias via Training

We can apply a Sobolev-norm-based filter to the transfer function:

$$\hat{\mathbf{y}}(s) = (1 + |s|)^{\beta} \mathbf{G}(is)\hat{\mathbf{u}}(s).$$

Intuitively, β reweighs the frequency domain.

- $\beta < 0 \Rightarrow$ low frequencies are weighted more, frequency bias is enhanced.
- $\beta > 0 \Rightarrow$ low frequencies are weighted less, frequency bias is diminished.

Surprisingly, β also affects the training.

The gradient of a generic loss $\mathcal{L}$ with respect to $\mathrm{Im}(a_j)$ satisfies

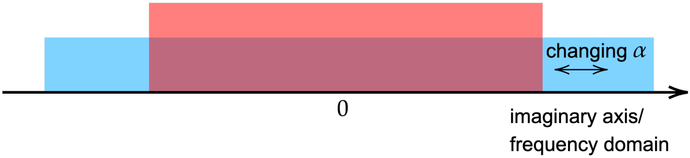
$$\frac{\partial \mathcal{L}}{\partial \mathrm{Im}(a_j)} = \int_{-\infty}^{\infty} \frac{\partial \mathcal{L}}{\partial \mathbf{G}(is)} \cdot K_j^{(\beta)}(s)\, ds,$$

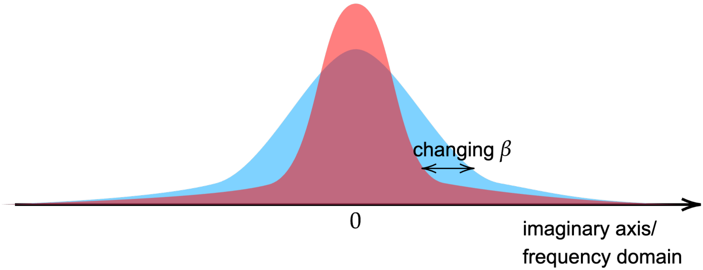
$$|K_j^{(\beta)}(s)| = \mathcal{O}\left(|s - \mathrm{Im}(a_j)|^{-2+\beta}\right).$$

Why Two Mechanisms?

Why Two Mechanisms?

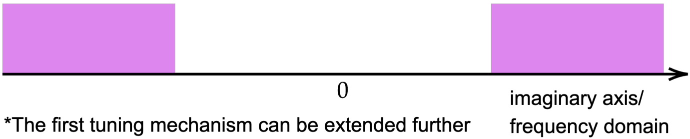
The hyperparameter α is a "hard" tuning strategy while β gives us a "soft" way.

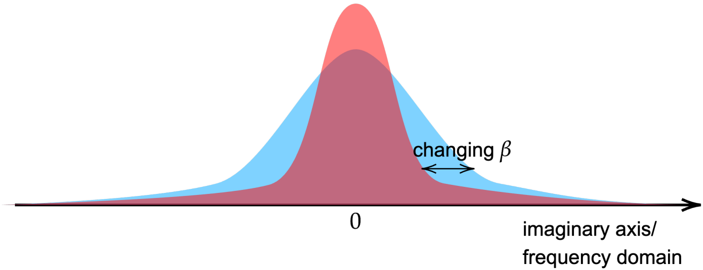




Why Two Mechanisms?

The hyperparameter α is a "hard" tuning strategy while β gives us a "soft" way.

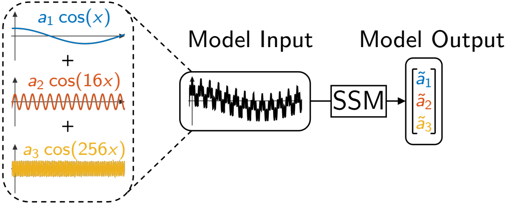




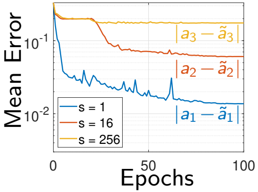
Some Examples of Tuning Frequency Bias

Some Examples of Tuning Frequency Bias

Problem Formulation

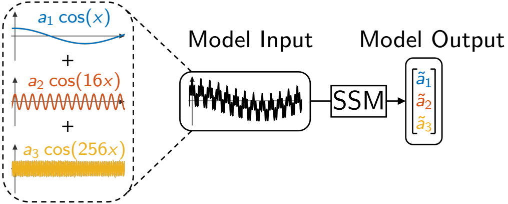


Results

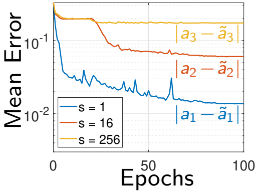


Some Examples of Tuning Frequency Bias

Problem Formulation

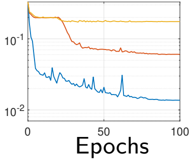


Results



Frequency bias is . . .

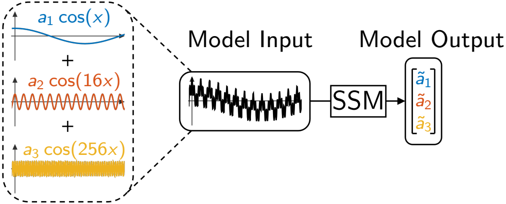
Default



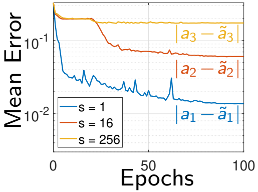
$(\alpha, \beta) = (1, 0)$

Some Examples of Tuning Frequency Bias

Problem Formulation

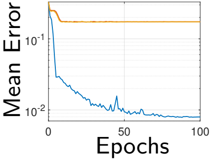


Results



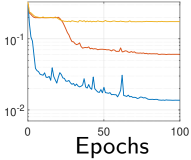
Frequency bias is . . .

Enhanced



$(\alpha, \beta) = (0.01, -1)$

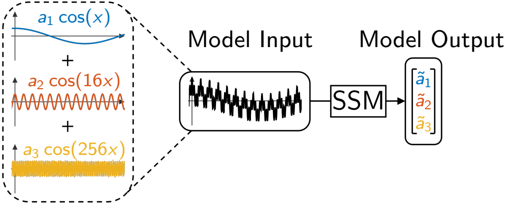
Default



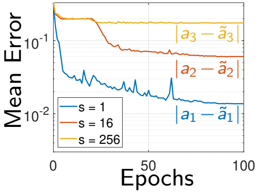
$(\alpha, \beta) = (1, 0)$

Some Examples of Tuning Frequency Bias

Problem Formulation

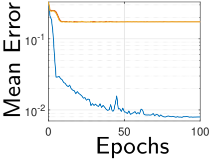


Results



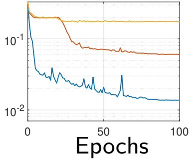
Frequency bias is ...

Enhanced



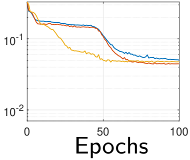
$(\alpha, \beta) = (0.01, -1)$

Default



$(\alpha, \beta) = (1, 0)$

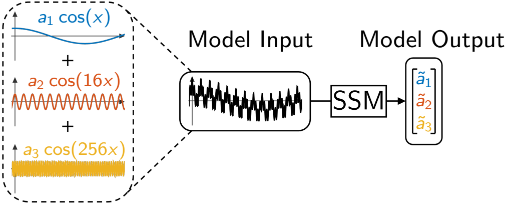
Counterbalanced



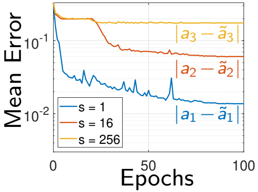
$(\alpha, \beta) = (10, 0.5)$

Some Examples of Tuning Frequency Bias

Problem Formulation

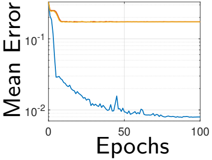


Results



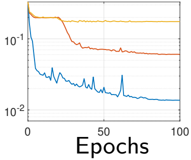
Frequency bias is ...

Enhanced



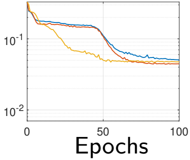
$(\alpha, \beta) = (0.01, -1)$

Default



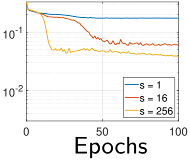
$(\alpha, \beta) = (1, 0)$

Counterbalanced



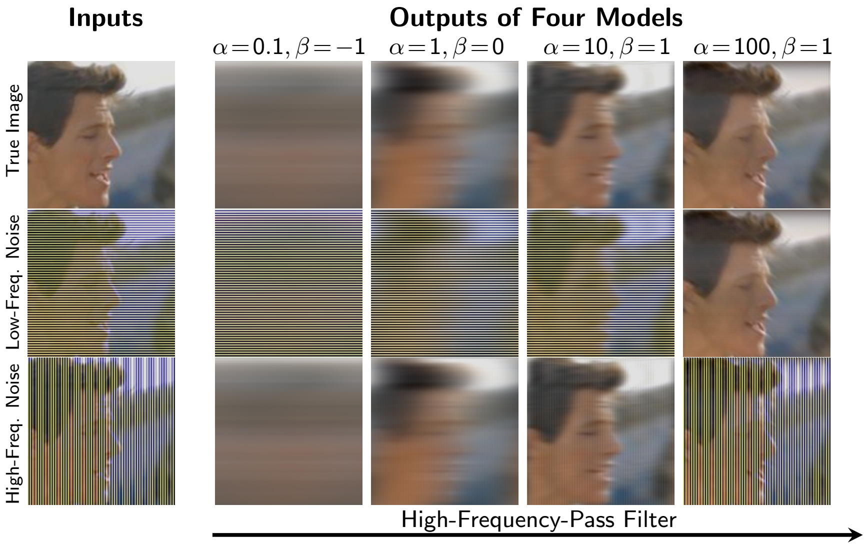
$(\alpha, \beta) = (10, 0.5)$

Reversed

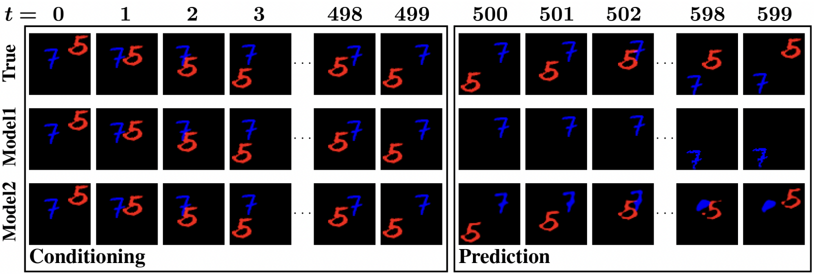


$(\alpha, \beta) = (100, 1)$

Some Examples of Tuning Frequency Bias



Some Examples of Tuning Frequency Bias



Conclusion

Conclusion

Conclusion:

Conclusion

Conclusion:

1. SSMs are linear RNNs that allow fast and numerically stable computation.

Conclusion

Conclusion:

1. SSMs are linear RNNs that allow fast and numerically stable computation.
2. Hankel singular values explain the success or failure of an SSM. HOPE gives a more robust parameterization.

Conclusion

Conclusion:

1. SSMs are linear RNNs that allow fast and numerically stable computation.
2. Hankel singular values explain the success or failure of an SSM. HOPE gives a more robust parameterization.
3. Frequency bias helps avoid overgeneralization but also prevents us from learning high-frequency information. Consider changing the hyperparameters α and β to tune frequency bias.

Conclusion

Conclusion:

1. SSMs are linear RNNs that allow fast and numerically stable computation.
2. Hankel singular values explain the success or failure of an SSM. HOPE gives a more robust parameterization.
3. Frequency bias helps avoid overgeneralization but also prevents us from learning high-frequency information. Consider changing the hyperparameters α and β to tune frequency bias.

Future Work:

Conclusion

Conclusion:

1. SSMs are linear RNNs that allow fast and numerically stable computation.
2. Hankel singular values explain the success or failure of an SSM. HOPE gives a more robust parameterization.
3. Frequency bias helps avoid overgeneralization but also prevents us from learning high-frequency information. Consider changing the hyperparameters α and β to tune frequency bias.

Future Work:

1. How do the real and the imaginary story interact?

Conclusion

Conclusion:

1. SSMs are linear RNNs that allow fast and numerically stable computation.
2. Hankel singular values explain the success or failure of an SSM. HOPE gives a more robust parameterization.
3. Frequency bias helps avoid overgeneralization but also prevents us from learning high-frequency information. Consider changing the hyperparameters α and β to tune frequency bias.

Future Work:

1. How do the real and the imaginary story interact?
2. Controls in SSMs.

Conclusion

Conclusion:

1. SSMs are linear RNNs that allow fast and numerically stable computation.
2. Hankel singular values explain the success or failure of an SSM. HOPE gives a more robust parameterization.
3. Frequency bias helps avoid overgeneralization but also prevents us from learning high-frequency information. Consider changing the hyperparameters α and β to tune frequency bias.

Future Work:

1. How do the real and the imaginary story interact?
2. Controls in SSMs.
3. SSMs for GenAI.