

Advanced Methods in Text Analytics

State Space Models



An Alternative to Transformers (1)

- Transformer-based models dominate NLP
 - Most applications tackled with learned representations
 - Learned representations obtained with transformers
 - Great, right?
 - Except for quadratic cost on input sequence length
 - Severely limits size of context fed to models
 - Many efforts to reduce cost, e.g.
 - [FlashAttention \(2022\)](#)
 - Recent efforts to get infinite context, e.g.
 - [TransformerFam \(2024\)](#),
 - [Leave No Context](#)
-
- A bar chart comparing the context lengths of various Large Language Models (LLMs). The y-axis represents the number of tokens, ranging from 0 to 100,000 in increments of 25,000. The x-axis lists several models, grouped by company. Hugging Face models are shown at the top with very long context lengths (near 100,000 tokens), while OpenAI, Meta AI, and Cohere models are shown below with much shorter context lengths (under 25,000 tokens).
- | Company | Model | Context Length (tokens) |
|--------------|--------------------------------|-------------------------|
| Hugging Face | bloom | ~100,000 |
| Hugging Face | flan-t5-xxl | ~100,000 |
| Hugging Face | cassini-4-pythia-12b-epoch-3.5 | ~100,000 |
| Hugging Face | cassini-3t-1-pythia-12b | ~100,000 |
| Hugging Face | santacoder | ~100,000 |
| Hugging Face | gpt-neox-20b | ~100,000 |
| Hugging Face | text-nada-001 | ~100,000 |
| Hugging Face | text-babbage-001 | ~100,000 |
| Hugging Face | text-curie-001 | ~100,000 |
| OpenAI | llama7b-v2-chat | ~16,000 |
| Meta AI | llama13b-v2-chat | ~13,000 |
| Cohere | llama70b-v2-chat | ~10,000 |
| Other | command-nightly | ~8,000 |
| Other | gpt-3 | ~4,000 |

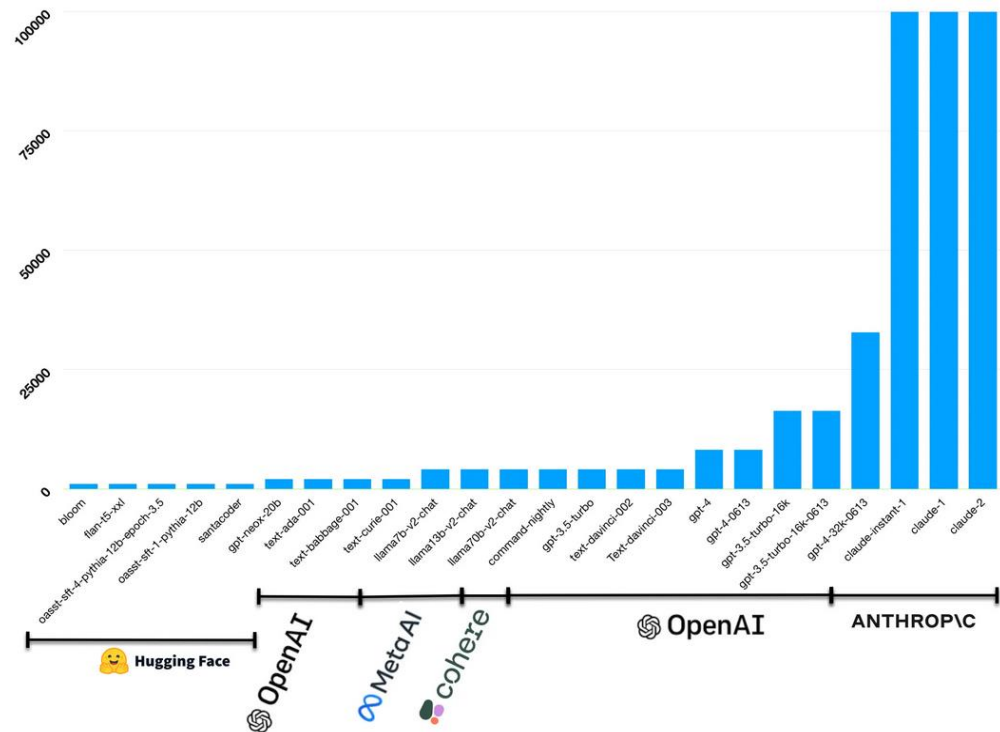


Image source

An Alternative to Transformers (2)

- Enter: state-space models
 - Approach from [control theory](#)
 - They are sequence models that do not use self-attention
 - Great at modeling very long-range sequences (e.g. 1M tokens)
- Today:
 - **State space models**, including S4 model (precursor to Mamba), introduced structure to state space models for efficiency, reached SOTA in long range dependency tasks
 - Relation to **RNNs** (you already know more about this than you think)
 - The **Mamba architecture**, recent state space model, showed competitive performance in causal language modeling when compared with LLMs twice its size (using less memory and compute)
- **Warning:**
 - Much of this comes from a **research area** that most NLP researchers (myself included) are **likely unfamiliar** with
 - **Some things** will be discussed in **detail**, **others** will be kept at **high-level**
 - As in the literature, many things illustrated with equations, not images

Outline

1. State Space Models
2. The Mamba Architecture

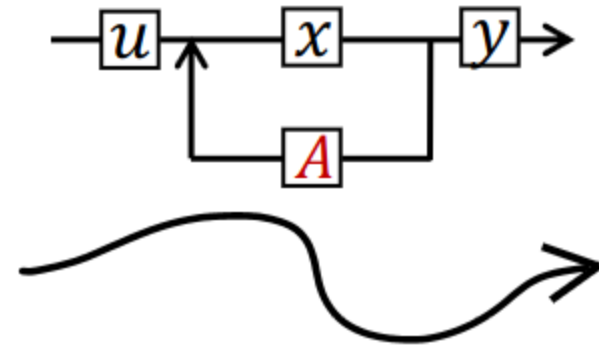
State Space Models

Continuous State Space Models (1)

- **Long range dependencies:** big challenge in sequence modelling
 - Materialized by **Long Range Arena** Benchmark ([Tay et al. 2021](#))
 - Wide range of tasks requiring processing between 1K and 16K tokens
- Most families of sequence models have variants to tackle this challenge
 - E.g. Lipschitz RNNs ([Erichson et al. 2021](#)), FlashAttention-2 ([Dao et al. 2023](#))
- Family of models that excel at this: **state space models (SSMs)**
- A ***continuous-time* latent state model** does the following:
 - 1. Maps 1-dimensional signal $x(t)$ to n-dimensional latent state $h(t)$
 - 2. Projects latent state (plus input signal) to 1-dimensional output signal $y(t)$
- Concretely:
 - 1. $dh(t)/dt = Ah(t) + Bx(t)$
 - 2. $y(t) = Ch(t) + Dx(t)$
- Here, **A , B , C , D** are learned parameters and t is time (usually $D = 0$ for simplicity because term $Dx(t)$ is seen as "skip connection" between input and output, easy to compute)

Continuous State Space Models (2)

- Let's note a few things about SSMs:
 - $\frac{dh(t)}{dt} = \mathbf{A}h(t) + \mathbf{B}x(t)$
 - $y(t) = \mathbf{C}h(t) + \mathbf{D}x(t)$
- The 1st equation is differential
 - Describes how \mathbf{h} changes over time
- The 2nd equation is **signal-to-signal model**
- Both $\frac{dh(t)}{dy}$ and $y(t)$ fully described by current input signal and hidden state
 - I.e. a **Markov decision process**
 - Ideally, state encodes all past history, enables us to determine future output $y(t)$
- Great, but this is NLP!
 - What are we supposed to do we these *continuous* SSMs?**
 - We are all about processing discrete sequences here, right?
- Yes, so **let's discretize SSMs**



$$\dot{x} = \mathbf{A}x + \mathbf{B}u$$

$$y = \mathbf{C}x + \mathbf{D}u$$

**Continuous
State Space**

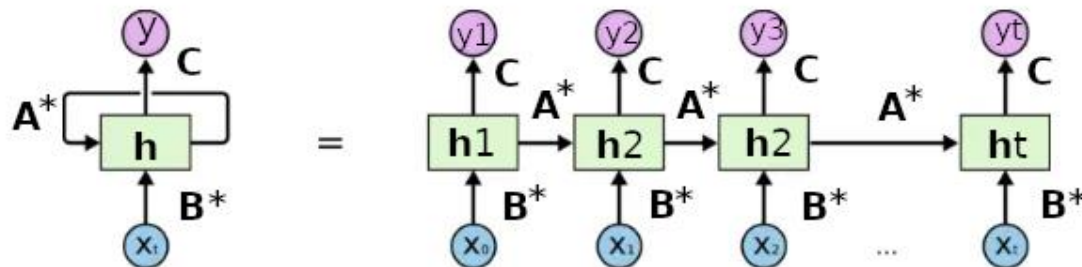
[Gu et al. 2022](#)

Discrete State Space Models (1)

- There are several ways to **discretize a continuous signal**
 - Think of it as **sampling a continuous signal at fixed time intervals**
 - Different SSMs use different approaches, each with different properties
- Simple way to get an *intuition for discretization*: [the Euler method](#)
- Let's start with our continuous model:
 - $\frac{dh(t)}{dt} = \mathbf{A}h(t) + \mathbf{B}x(t)$ (1)
 - $y(t) = \mathbf{C}h(t) + \mathbf{D}x(t)$ (2)
- 1st, we clean up notation a bit: $x_t = x(t)$, $\mathbf{h}_t = \mathbf{h}(t)$ and $\mathbf{h}_{t+1} = \mathbf{h}(t + \Delta)$
- 2nd, by definition of a derivative: $\frac{dh(t)}{dy} = (\mathbf{h}(t + \Delta) - \mathbf{h}(t)) / \Delta$ (3)
- Then, by plugging (3) in (1), we have:
 - $\mathbf{h}_{t+1} - \mathbf{h}_t = \Delta(\mathbf{A}\mathbf{h}_t + \mathbf{B}x_t)$
 - $\mathbf{h}_{t+1} = \Delta\mathbf{A}\mathbf{h}_t + \Delta\mathbf{B}x_t + \mathbf{h}_t$
 - $\mathbf{h}_{t+1} = \mathbf{h}_t(\Delta\mathbf{A} + \mathbf{I}) + \Delta\mathbf{B}x_t$
 - $\mathbf{h}_{t+1} = (\mathbf{I} + \Delta\mathbf{A})\mathbf{h}_t + (\Delta\mathbf{B})x_t$

Discrete State Space Models (2)

- If we now set $\mathbf{A}^* = (\mathbf{I} + \Delta\mathbf{A})$, $\mathbf{B}^* = \Delta\mathbf{B}$ and set $\mathbf{D} = \mathbf{0}$, we have:
 - $\mathbf{h}_{t+1} = \mathbf{A}^*\mathbf{h}_t + \mathbf{B}^*\mathbf{x}_t$ (1)
 - $\mathbf{y}_t = \mathbf{C}\mathbf{h}_t$ (2)
- Does that look familiar?
 - I hope we haven't forgotten! It looks like an RNN



An unrolled recurrent neural network.

[Image source](#)

- Discrete SSMs are essentially linear RNNs (i.e. seq2seq models)
 - I.e. RNN with activation $f = \text{I}$ when computing \mathbf{h} , i.e. $\mathbf{h}_{t+1} = f(\mathbf{A}^*\mathbf{h}_t + \mathbf{B}^*\mathbf{x}_t)$
 - Computing \mathbf{A}^* , \mathbf{B}^* (discretization) can be seen as first step in compute graph

Discrete State Space Models (3)

- As mentioned, different SSMs use different approaches for discretization
 - None actually uses the Euler method
- E.g. [S4](#) uses a bilinear method:
 - $\mathbf{A}^* = (\mathbf{I} - \Delta/2 \mathbf{A})^{-1}(\mathbf{I} + \Delta/2 \mathbf{A})$
 - $\mathbf{B}^* = (\mathbf{I} - \Delta/2 \mathbf{A})^{-1} \Delta \mathbf{B}$
- [Mamba](#) uses a process called [zero-order hold](#):
 - $\mathbf{A}^* = \exp(\Delta \mathbf{A})$
 - $\mathbf{B}^* = (\Delta \mathbf{A})^{-1}(\exp(\Delta \mathbf{A}) - \mathbf{I}) \Delta \mathbf{B}$
- Such models are thus parameterized by $\boldsymbol{\theta} = [\Delta, \mathbf{A}, \mathbf{B}, \mathbf{C}]$
- Ok, fine, so we are back to (linear) RNNs. Is that it?
- Not quite, as **RNNs fell out of favor for two specific reasons**
 - 1. Difficulty modeling long range dependencies
 - 2. Slow to train due to sequential computation of hidden states
- **SSMs need to deal with these issues** to be successful
 - Let's have a high-level look at how SSMs addressed these "RNN challenges"

Modeling Long Sequences with SSMs

- **HiPPO framework** introduced by [Gu et al. 2020](#)
 - **H**igh-order **P**olynomial **P**rojection **O**perators
 - Math-heavy work (described as "a bit of magic" by [some sources](#))
- Essentially, framework that proposes a **principled way of initializing matrix A** so state $h(t)$ can memorize input sequence x
 - Specifically, a complex matrix in the following upper triangular form:

$$(\text{HiPPO Matrix}) \quad A_{nk} = - \begin{cases} (2n+1)^{1/2}(2k+1)^{1/2} & \text{if } n > k \\ n+1 & \text{if } n = k \\ 0 & \text{if } n < k \end{cases}$$

- More generally, they formalized notion of *memory as online function approximation* (based on [approximation theory](#) and [signal processing](#))
 - The framework derived [GRUs](#) and [LMUs](#) from first principles
- Their results put SSMs on the map, reported **massive improvements**
 - E.g. from 68% to 98% on sequential MNIST benchmark
- Still, SSMs were expensive to train compared to Transformers
 - One RNN challenge down, one to go

Training SSMs Efficiently

- Important property of SSMs: **linear time invariance** (LTI)
 - Well known connection with convolutions (yet no citation in papers!)
- In short, because computing \mathbf{h}_{t+1} is linear, we can unroll SSMs as follows:
 - $\mathbf{h}_0 = \mathbf{B}^* \mathbf{x}_0$ $\mathbf{h}_1 = \mathbf{A}^* \mathbf{B}^* \mathbf{x}_0 + \mathbf{B}^* \mathbf{x}_1$ $\mathbf{h}_2 = \mathbf{A}^{*2} \mathbf{B}^* \mathbf{x}_0 + \mathbf{A}^* \mathbf{B}^* \mathbf{x}_1 + \mathbf{B}^* \mathbf{x}_2$
 - $\mathbf{y}_0 = \mathbf{C} \mathbf{B}^* \mathbf{x}_0$ $\mathbf{y}_1 = \mathbf{C} \mathbf{A}^* \mathbf{B}^* \mathbf{x}_0 + \mathbf{C} \mathbf{B}^* \mathbf{x}_1$ $\mathbf{y}_2 = \mathbf{C} \mathbf{A}^{*2} \mathbf{B}^* \mathbf{x}_0 + \mathbf{C} \mathbf{A}^* \mathbf{B}^* \mathbf{x}_1 + \mathbf{C} \mathbf{B}^* \mathbf{x}_2$
 - and so on, where initial state $\mathbf{h}_{-1} = \mathbf{0}$.
- Thus, for input sequence \mathbf{x} , the entire output sequence is $\mathbf{y} = \mathbf{K} \odot \mathbf{x}$ where \odot is the (discrete) **convolution** operation and \mathbf{K} is the following (giant) convolution filter:
 - $\mathbf{K} = (\mathbf{C} \mathbf{B}^*, \mathbf{C} \mathbf{A}^* \mathbf{B}^*, \mathbf{C} \mathbf{A}^{*2} \mathbf{B}^*, \dots, \mathbf{C} \mathbf{A}^{*|\mathbf{x}|-1} \mathbf{B}^*)$
- I.e., **we can compute entire forward pass as single global convolution!**
 - Can be computed efficiently with [Fast-Fourier Transforms](#) (FFTs) once \mathbf{K} is known
 - But computing \mathbf{K} is expensive (repeated matrix multiplication by \mathbf{A}^*)
- Achieving this in linear time was a main contribution of the S4 model

Structured State Space Models

- **S4**: structured state space sequence models ([Gu et al. 2022](#))
 - Again, math-heavy paper, only high-level discussion here
- **Main contribution**:
 - Computing global kernel \mathbf{K} in linear time (normally quadratic)
- They **enforce structure** to parameter matrix \mathbf{A} in SSMs
 - \mathbf{A} is special form of **diagonal matrix**: diagonalizable by normal matrices
 - Work far from trivial, as matrix \mathbf{A} had to retain the properties of the HiPPO matrix that accounts for the success of SSMs at long range dependencies
- An S4 layer was proposed to construct deep S4 models
 - **S4 block**: SSM seq2seq model + dropout + non-linearity + linear projection
 - Since S4 processes single scalar, **S4 block contains H copies of S4 model**
 - Then, each block processes inputs of size (batch size, input length, hidden size), same as RNNs, Transformers (but S4 is linear in input length!)
 - Can be **computed as convolution** (training) **or autoregressively** (inference) (nicely shown in code [here](#))
- **So, with S4 we get efficient SSM training without performance loss**
 - Now let's recap SSMs vs RNNs before we look at Mamba

SSMs vs RNNs

- While similar, there are **important differences between SSMs and RNNs**
 - 1. SSMs (and linear RNNs) can be efficiently parallelized (global convolution)
 - 2. SSMs are "linear RNNs" but with special requirements for how to compute some parameters (discretization)
 - 3. SSMs are complex valued and initialized according to the HiPPO theory
- Also, note that while SSM matrices \mathbf{A}^* , \mathbf{B}^* are equivalent to RNN parameters, they actually share parameters themselves (usually Δ and \mathbf{A})
- In short, **these differences are important**, account for success of SSMs!
- After S4's success, **recent work "resurrected RNNs"** ([Orvieto et al. 2023](#))
 - Started with a linear RNN, "ablated" their way to reach S4's performance
 - Specifically, they introduced **diagonalization**, **special initialization** and **parameterization**, as well as **normalization** to RNNs
 - Insightful in terms of where success of SSMs comes from
- Having "covered" how both RNN challenges were addressed by SSMs, **we can now discuss Mamba!**
 - This because the model is an extension of the S4 architecture

The Mamba Architecture

Motivation

- Perspective proposed by authors:
 - *"A fundamental problem of sequence models is compressing context into a small state"*
- For example, **self-attention** is:
 - **Effective:** does not compress context at all, stores it entirely (KV cache in causal LMs)
 - **Inefficient:** KV cache takes linear space in input sequence length
- Conversely, **RNNs** are:
 - **Efficient:** compress context in hidden state h (size usually much smaller than the long range sequences we want to model with SSMs)
 - **Ineffective:** especially for long sequences, as performance depends on how well they compress context (though the "RNN resurrection" paper improved considerably on this thanks to research in SSMs)
- In short, **efficient vs effective trade-off** summarized by how well a model compresses their state
 - Proposed **fundamental principle** for building sequence models: **selectivity**

Improving SSMs with Selection

- **Selectivity:** context-aware ability to focus on or filter out inputs into a sequential state
 - Sounds awfully familiar, right?
- In other words, an **attention-like mechanism** that **controls how information propagates** or **interacts** between elements of a sequence
 - Is it really an attention-like mechanism?
- Yes! The **main Mamba design innovation** is allowing **more access to input sequence** by letting the **SSM parameters** be **input-dependent**
 - Recall attention on RNNs was basically: *Hey, why don't we just access the input at each inference step instead of just interacting with the hidden state?*
 - Mamba is the SSM that asks the same question (likely inspired by the success of the Transformer).
- Specifically, some parameters became functions of input sequence \mathbf{x}
 - Matrix $\mathbf{B} = \mathbf{W}_B \mathbf{x}$, matrix $\mathbf{C} = \mathbf{W}_C \mathbf{x}$ where \mathbf{W}_B , \mathbf{W}_C are learned projections
- **Problem:** this change means we lose the ability to train SSMs efficiently
 - Let's see why

Improving SSMs with Selection?

- Recall efficient training of SSMs:
 - Given input sequence \mathbf{x} , entire output sequence is $\mathbf{y} = \mathbf{K} \odot \mathbf{x}$ where \odot is (discrete) convolution operation and \mathbf{K} is the following (giant) convolution kernel: $\mathbf{K} = (\mathbf{CB}^*, \mathbf{CA}^*\mathbf{B}^*, \mathbf{CA}^{*2}\mathbf{B}^*, \dots, \mathbf{CA}^{*|\mathbf{x}|-1}\mathbf{B}^*)$
 - Efficient! Compute kernel elements separately, apply [convolution theorem](#)
- Now, note the following:
 - \mathbf{h}_t depends on \mathbf{h}_{t+1} because $\mathbf{h}_{t+1} = \mathbf{A}^*\mathbf{h}_t + \mathbf{B}^*\mathbf{x}_t$
 - Since $\mathbf{B} = f(\mathbf{x})$, $\mathbf{B}^* = f(\mathbf{x})$ because \mathbf{B}^* is a function (discretized form) of \mathbf{B}
 - For $t = 1$, $\mathbf{x} = [\mathbf{x}_1]$, for $t = 2$, $\mathbf{x} = [\mathbf{x}_1, \mathbf{x}_2]$, for $t = n$, $\mathbf{x} = [\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n]$
- So, \mathbf{B}^* is now time dependent, i.e. \mathbf{B}^*_t
 - Thus, a more accurate notation now is: $\mathbf{h}_{t+1} = \mathbf{A}^*\mathbf{h}_t + \mathbf{B}^*_t\mathbf{x}_t$ since \mathbf{B}^* is a function of \mathbf{x} , which changes per time step (autoregressive models)
- In other words, **hidden states now need to be computed in sequence**
 - We lost one of the two improvements SSMs had over RNNs
 - So, another main **contribution from Mamba: make training efficient again**

Hardware-Aware Selectivity

- To regain training efficiency, the authors did two things:
 - 1. Switch from convolution to a [scan](#) operation
 - 2. Implement **scan in a GPU-optimized way**
- **Scan:** otherwise known as a cumulative sum
 - E.g. given sequence $x = [3, 2, 6]$, output is sequence $y_1 = 3, y_2 = 5, y_3 = 11$
 - Can be parallelized to be computed in $\log n$ (nicely shown [here](#))
 - Already used for efficiently training linear RNNs ([Orvieto et al. 2023](#))
- So, in short: **what is Mamba?**
 - An **S4 model** (structured SSM) with **Selectivity**
 - An S4 model trained with the **Scan** operation
 - **Hence**, authors described it as an **S6 model**
- **Transformers vs Mamba:** [overall training costs](#) for sequence of length n
 - **Memory:** $O(n)$ vs $O(1)$ (KV cache vs hidden state)
 - **Runtime:** $O(n^2)$ vs $O(n)$ (self-attention vs GPU-optimized SSM)
- In other words, massive improvements!
- If performance on par with LLMs, we may have a new king

The Mamba Block

- As with the Transformer, we have a **Mamba block**

- Inter-token communication:**

- Transformer: self-attention
- Mamba: SSM

- Intra-token computation:**

- Transformer: MLP
- Mamba: MLP

- Efficiency:**

- Transformer: linear attention
- Mamba: linear RNN

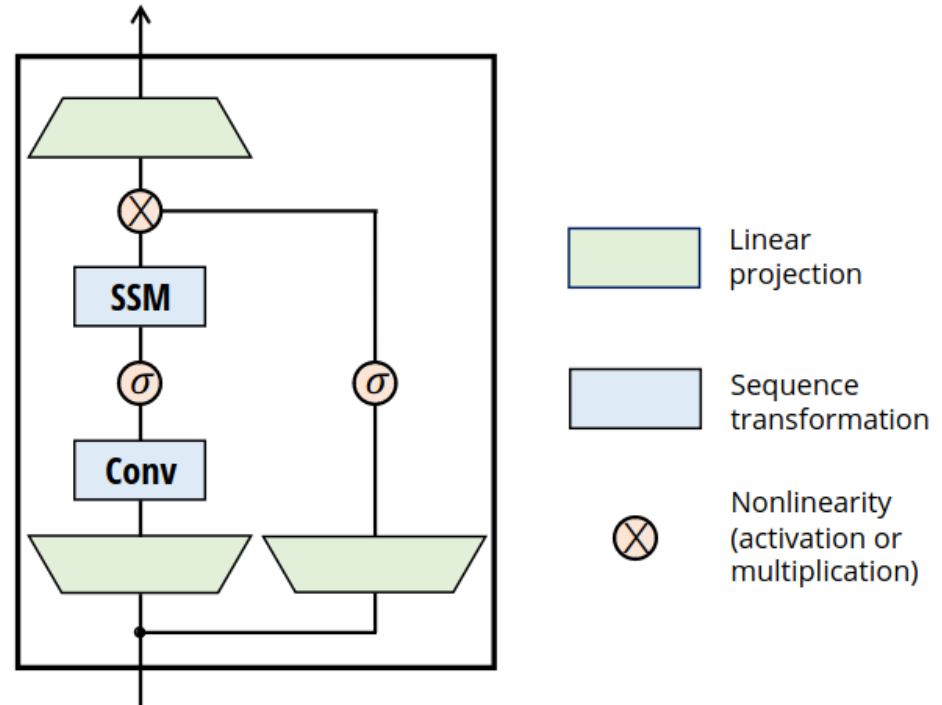
- Expressivity:**

- Transformer: non-linearity between MLP projections
- Mamba: non-linearity between MLP projections

- In practice:**

- Up-projection by a factor of 2 (compared to usual 4 in LLMs)

- [Silu/Swish](#) activations, optional layer normalization before/after block



[Gu et al. 2023](#)

Mamba vs LLMs

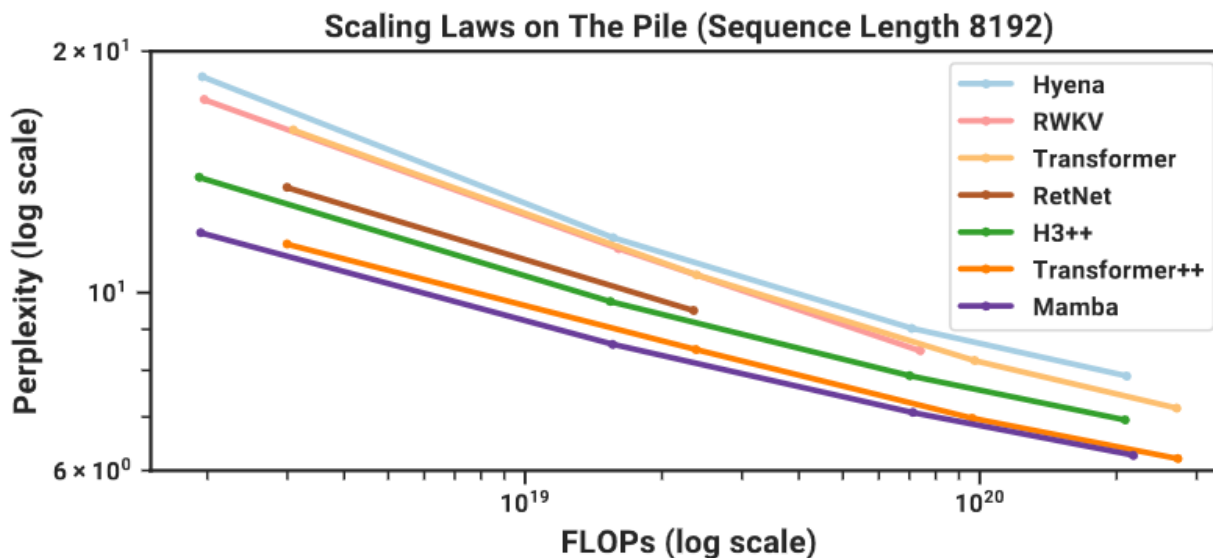
- Some **results** on **downstream tasks** (with **5x faster inference** compared to transformer-based LLMs)

MODEL	TOKEN.	PILE PPL ↓	LAMBADA PPL ↓	LAMBADA ACC ↑	HELLASWAC ACC ↑
GPT-Neo 2.7B	GPT2	—	5.63	62.2	55.8
Hybrid H3-2.7B	GPT2	—	7.92	55.7	59.7
OPT-2.7B	OPT	—	5.12	63.6	60.6
Pythia-2.8B	NeoX	6.73	5.04	64.7	59.3
RWKV-3B	NeoX	7.00	5.24	63.9	59.6
Mamba-2.8B	NeoX	6.22	4.23	69.2	66.1
GPT-J-6B	GPT2	—	4.10	68.3	66.3
OPT-6.7B	OPT	—	4.25	67.7	67.2
Pythia-6.9B	NeoX	6.51	4.45	67.1	64.0
RWKV-7.4B	NeoX	6.31	4.38	67.2	65.5

- Some **concerns** from the community: [Gu et al. 2023](#)
 - Are baselines strong enough? E.g. no latest OpenAI models
 - More importantly, **why no comparison with larger LLMs?**

Scaling Laws

- Authors also provided scaling laws on all tested tasks (here only LM)
 - They followed the [Chinchilla protocol](#)
 - Roughly, statistical model that measures loss as function of model size



[Gu et al. 2023](#)

- "First attention-free model to match performance of very strong Transformer recipe (Transformer++)"** (with cheaper memory/compute!)
 - Transformer++ recipe: Rotary, SwiGLU, MLP, RMSProp, no linear bias, higher learning rate (PaLM, LLaMA)

A Different Paradigm (1)

- Nice discussion on a potential paradigm shift by [Kola Ayonrinde](#)
 - Specifically about using pre-trained causal language models (CLMs)
- **Sources of information** in transformer-based CLMs **at inference time?**
 - **Training data**, i.e. pre-trained weights (long-term memory, but compressed)
 - **In-context data**, i.e. prompt (short-term memory, must be read every time)
- Do we have **selectivity when using such CLMs?**
 - Yes, via **prompting** (what to leave in or out), **RAG**, etc.
 - This because entire prompt is accessed (attention) to predict next tokens
- What about when using **SSM-based CLMs like Mamba?**
 - Training data is there in the same way, i.e. compressed long-term memory
 - Selectivity is a core design of the system as it reads any input
 - But **in-context data is also long-term now!** Why?
- Because **we can store the last hidden state, plug it in later as needed!**
 - Transformers not designed to have a state or representation plugged in
 - They must instead "read" the entire input to produce contextualized vectors

A Different Paradigm (2)

- This is **essentially a new form of prompting**
 - Swapping hidden states as needed
 - A modularity akin to swapping LoRA modules
- Remember: **SSMs are very good at long range dependencies**
 - We could potentially read entire set of textbooks, store hidden state h_n
 - Then, at inference time, use h_n as initial state of our prompt
- **Question:** can transformers' contextual embeddings also act as states?
 - There is already [some work](#) on this, so maybe...
- And with that, our discussion on SMMs comes to an end
 - So, let's summarize!

Summary

- **State Space Models (SSMs):**
 - **Sequence models similar to linear RNNs, but with key differences**
 - Differences make them **very good at long range dependencies**
 - This ability comes from **principled design choices** (HiPPO framework)
- **Mamba Architecture:**
 - **SSM with attention-like mechanism** for selectively using input sequence
 - Performs **competitively with LLMs twice its size**
 - Scaling laws show promise for scaling up the model
- **SSMs vs Transformers**
 - **SSMs much more efficient**, memory: $O(1)$ vs $O(n)$, compute: $O(n)$ vs $O(n^2)$
 - But **no results on Mamba vs largest LLMs yet**, e.g. 70B or larger

References

- [Efficiently Modeling Long Sequences with Structured State Spaces](#) by Gu et al, 2022
- [The Annotated S4](#) by Sasha Rush and Sidd Karamcheti
- [Resurrecting Recurrent Neural Networks for Long Sequences](#) by Orvieto et al. 2023
- [Mamba: Linear-Time Sequence Modeling with Selective State Spaces](#) by Gu and Dao, 2023
- [Mamba Explained](#) by Kola Ayonrinde
- [Mamba from Scratch](#) by Algorithmic Simplicity
- References linked in corresponding slides