Summary of Models

Jaehyeon Shin

1. Convolutional Neural Network

I. Introduction

a. CNNs had been major breakthroughs in image labeling, object detection, scene classification, areas. Specifically, Convolutional neural network. This work focuses on identifying the best network for this purpose. Feature extraction is a key step of such algorithms. Feature extraction from images involves extracting a minimal set of features containing a high amount of object or scene information from low-level image pixel values, therefore, capturing the difference among the object categories involved.

b. A CNN, known for local connectivity of neurons, weight sharing, and down-sampling, is a deep feed-forward multi-layered hierarchical network inspired by the receptive field mechanism in biology. CNNs constitute one such class of models. Their capacity can be controlled by varying their depth and breadth. Thus, compared to standard feedforward neural networks with similarly-sized layers, CNNs have much fewer connections and parameters and so they are easier to train. Convolutional networks are a specialized kind of neural networks for processing data that has a known grid-like topology. Convolutional networks have been tremendously successful in practical applications.

II. Basic CNN components

Although there are numerous variations of CNN models, the overall architecture is essentially the same and adheres to a fixed paradigm. The first half of the network comprises of a number of convolution and pooling layers stacked alternately to form a feature extractor. Fully connected layers are used in combination with activation functions to execute tasks such as classification or regression on the extracted features. To maximize CNN performance, various regulatory units like batch normalization and drop out are also included in addition to various mapping functions.

a. Input Layer

- The first layer of each CNN used is ‘input layer’. This layer receives images, resizes them on for feature extraction.

b. Convolutional layer

- The convolution layer, which may extract various features from different local regions of the input data, is composed of a collection of convolution kernels, with each neuron acting as a kernel. By convolving the original image with the filters and applying a nonlinear activation function to obtain new feature mappings, each feature mapping can be used as a class of extracted image features. To extract higher-level and more complete feature representations, the network model can stack multiple convolution layers.

- This layer’s weight-sharing technique allows multiple sets of features within an image to be retrieved by sliding a kernel with the same set of weights on the image, making it more efficient and effective for CNN parameters than fully connected networks. Furthermore, it also allows the network to have fewer neuron connections and a simpler network architecture, which facilitates the training of the network.

c. Pooling layer

- The neurons in the pooling layer are connected to the local receptive domains or their input layer, i.e., the convolution layer, and the local receptive domains of different neurons do not overlap.

- Pooling functions that are commonly used are max pooling and average pooling. Max pooling retains the most significant features by selecting the maximum value with in a region, while average pooling computes the mean. The mean value of the local response of the extracted feature mapping.

- The introduction of a pooling layer not only effectively compress the amount of data and parameters, reduces the feature map dimension, and minimizes overfitting, but also makes the network invariant to some small local morphological changes while having a larger perceptual filed.

d. Fully connected

- A fully connected layer is typically employed at the network's conclusion for classification. Each neuron in the fully connected layer is connected one by one with all the neurons in its preceding layers. Once the feature mapping obtained after several convolution and pooling operations is sufficient to recognize the features of the image. Generally, the CNN will pull the multiple feature mappings that are finally obtained at the end into a long vector and send to the fully connected layer. In addition, the fully connected layer can integrate local information that is class-distinctive in the convolution or pooling layers. As a result, This layer is the fully connected layers which takes the high-level filtered images and translate them into categories with labels.

e. Activation function

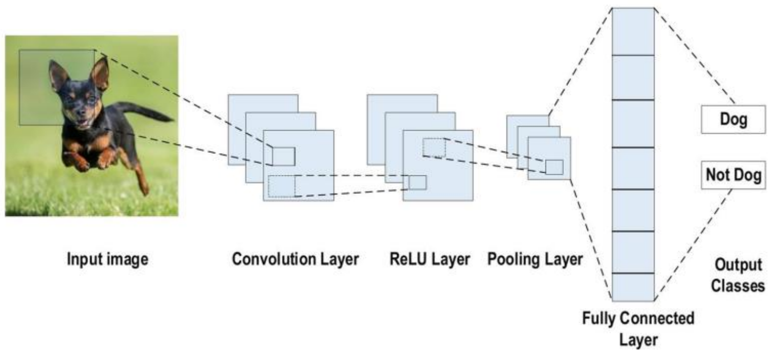
- An activation function is a different mathematical function that receives the filter's output. It plays an important role in neural networks, which strengthens the representational and learning capabilities of the network. Common functions include ReLU, sigmoid. For example, The ReLU layer swaps every negative number of the pooling layer with 0. This helps the CNN stay mathematically stable by keeping learned values from getting stuck near 0 or blowing up toward infinity.

f. Batch normalization

- Batch normalization is a neural network regularization technique that unifies the distribution of feature-map values by setting them to zero mean and unit variance. The BN layer helps to alleviate the problem of gradient vanishing and gradient explosion, improves the network's adaptability to different input data, speeds up the neural network's training process, and improves the network's generalization.

g. Dropout

- Dropout facilitates regularization in the network by randomly omitting some units or connections with a predetermined probability, which eventually enhances generalization.



[11]

III. Strength

a. The main benefit of CNN compared to its predecessors is that it automatically identifies the relevant features without any human supervision. CNNs have been extensively applied in a range of different fields, including computer vision, speech processing, Face Recognition, etc. Concurrently learning feature extraction layers and the classification layer causes the model model output to be highly organized and reliant on the extracted featured.

b. Convolution leverages three important ideas that can help improve a machine learning system: sparse interactions, parameter sharing and equivariant representation. Unlike conventional fully connected(FC) networks, shared weights and local connections in the CNN are employed to make full use of 2D input-data structures. Traditional neural network layers use matrix multiplication by a matrix of parameters with a separate parameter describing the interaction between each input unit and each output unit. This means that every output unit interacts with every input unit.

- This operation utilizes an extremely small number of parameters, which both simplifies the training process and speeds up the network. The weight sharing feature reduces the number of trainable network parameter and in turn helps the network to enhance generalization and to avoid overfitting.
- Convolutional network typically have sparce interactions. This is accomplished by making the kernel smaller than the input. This means that we need to store fewer parameters, which both reduces the memory requirements of the model and improves its statistical efficiency. It also means that computing the output requires fewer operations.
- Parameter sharing refers to using the same parameter for more than one function in a model. In a traditional neural net, each element of the weight matrix is used exactly once when computing the output of a layer. It is multiplied by one element of the input and then never revisited. As a synonym for parameter sharing, one can say that a network has tied weights, because the value of the weight applied to one input is tied to the value of a weight applied elsewhere. The parameter sharing used by the convolution operation means that rather than learning a separate set of parameters for every location, we learn only one set. This does not affect the runtime of forward propagation but it does further reduce the storage requirements of the model to k parameters. k is usually several orders of magnitude smaller than m.

IV. Drawback

a. Lack of interpretability

- Given that deep CNNs are black boxes, interpretation and explanation may be lacking. As a consequence it can be difficult to verify them at times. The solid theory of CNNs is still lacking. We don't know why and how it works essentially.

b. Resource Demands and Deployment Challenges

- Deep CNN models for mobile devices are difficult to implement due to the necessity of maintain model accuracy while taking into account the model's size and performance. CNN deep models, when evaluated, need plenty of memory to hold numerous parameters and highly time-consuming, making them unfeasible for deployment on resource-limited mobile platforms and other portable devices.

c. Complex Hyperparameter Tuning

- One major barrier for applying CNN on a new task is that it requires considerable skill and experience to select suitable hyperparameters such as the learning rate, kernel sizes of convolutional filters, the number of layer, etc. These hyper-parameter have internal dependencies which make them particularly expensive for tuning.

d. Coordinate Frame

- Convolutional networks recognize the image in terms of pixels which are arranged distinct patterns and don't understand them as component which are present in the image. The images as visualized by CNN do not have internal representation of components and their part-whole relationships.

e. Adversarial examples

- It is vulnerable to adversarial images If the CNN takes an image along with some noise, it recognizes the image as a completely different images. This vulnerability highlights a critical weakness in the robustness of CNNs against manipulated inputs.

2. Transformer

I. Introduction

a. The introduction of the Transformer model was a significant advancement aimed at overcoming the limitations of previous natural language processing (NLP) models. RNN is a class of neural networks designed to handle sequential data. Unlike traditional feedforward neural networks, RNNs have connections that form directed cycles, allowing them to maintain a memory of previous inputs. Recurrent neural network (RNNs) and long short-term memory networks(LSTMs) had strengths in handling sequential data but also faced several challenges.

- First, there was the difficulty in parallelization. RNNs and LSTMs process input sequence sequentially due to their reliance on the order of data. This characteristic made it difficult to process long sentences or sequences efficiently. which can be computationally expensive and time-consuming. This sequential processing makes it difficult to parallelize training and inference, limiting the scalability and efficiency of RNN-based models. The models are challenging for handling tasks that require understanding relationships between distant elements in the sequence.

- Second, the problem of long-term dependencies was a significant issue. Although RNNs and LSTMs are theoretically capable of capturing long-term dependencies in sequences, in practice, they often struggled with maintaining information over long sequences. As a result, the performance in tasks requiring long-term context often degraded as the sequence length increased.

- Furthermore, existing problem of RNN is gradient vanishing and gradient exploding. One of the significant challenges with RNNs is the vanishing and exploding gradient problem. During training, the gradients used to update the network's weights can become very small or very large, making it difficult for the network to learn effectively.

b. Transformers have been successfully applied to sequential, auto-regressive tasks despite being feed forward networks. Unlike recurrent neural networks, Transformers use attention to capture temporal relations while processing input tokens in parallel. Unlike previously dominant RNN-based model like Long Short-Term Memory, Transformers do not rely on recurrence. Instead, It use a self-attention mechanism that allows for more efficient processing of sequential data. And a self-attention mechanism comprehends the contextual relationships within sequential data.

II. Main component

a. Scaled dot product attention

- The scaled dot product attention function is computed as follows:

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

  d_k is the dimensionality of keys. The matrices Q and K represent the Query and Key vectors respectively, each with dimensions d_k. The matrix V represents the values vector.

- When the Query matrix is multiplied by the Key Matrix, we obtain the Attention score matrix. The Attention score is calculated by multiplying the Query and Key matrices, and since this operation represents the similarity between the matrices, the Attention score is essentially a similarity matrix the similarity between the Query and Key

- When performing the scale operation in Scaled Dot-product Attention, the scaling factor is determined by the dimensionality of the Key vector. For large values of d_k, the dot products grow large in magnitude. This poses a problem when applying the Softmax function afterward. This can lead to an issue known as vanishing gradient, where the gradient becomes too small to contribute effectively to learning, causing the learning process to slow down significantly. To prevent this, the original values are scaled down to keep the values within a more manageable range before applying Softmax. After this step, the Attention scores are normalized to a range between 0 and 1 using the Softmax function.

- The softmax score determines how much each word will be expressed at this position. And the softmax output is used to weigh the value vectors, highlighting relevant information and filtering out less important data. Finally, sum up the weighted value vector which produces the output of the self-attention layer at this point.

b. Multi-Head attention

- Instead of performing a single attention function with d_{model}-dimensional keys, values, and queries. Multi-Head Attention projects the queries, keys, and values linearly into different subspaces multiple times, each with different learned linear projections to d_k, d_k, d_v. Instead of performing a single attention operation, it splits them into h distinct subspaces. Attention is then computed in parallel for each of these subspaces. These outputs are then concatenated and undergo another linear projection to produce the final result.

- The advantage of Multi-Head attention is that it allows the model to capture and attend to different types of information from different subspaces simultaneously.

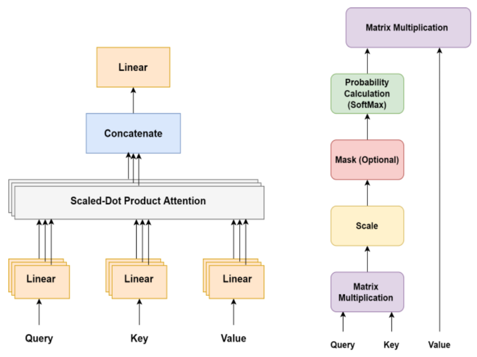
  The mathematical formulation for Multi-Head Attention can be described as:

$$MultiHead(Q, K, V) = Concat(head_1, \dots, head_h)W^O$$

$$Where\ head_i = Attention(QW_i^{\ Q}, KW_i^{\ K}, QV^V)$$

  Where the projections are parameter matrices:

$$W_i^{\ Q} \in \mathbb{R}^{d_{model} \times d_k}, W_i^{\ K} \in \mathbb{R}^{d_{model} \times d_k}, W_i^{\ V} \in \mathbb{R}^{d_{model} \times d_v}$$



[15]

c. Masked Multi-Head Attention layer

- These layers use the scaled dot product and Mask Operations to exclude future predictions and consider only previous outputs. The self-attention is restricted such that queries at each position can only attend to all key-value pairs up to and including that position. To enable parallel training, this is typically done by applying a mask function to the unnormalized attention matrix $A = \exp\left(\frac{QK^T}{\sqrt{D_K}}\right)$, where the illegal positions are masked out by setting $A_{ij} = -\infty$ (where i < j)

d. Positional Encoding

- Unlike recurrent neural networks or convolutional neural networks, Transformers lack inherent awareness of token order. Positional encoding provides a relative position for each token or word in a sequence. we use the sine and cosine functions to generate a unique vector for each position in the sequence. The output of sine and cosine is in [-1, 1], which is normalized. It outputs a unique encoding for each time-step (word's position in a sentence) Distance between any two time-steps is consistent across sentences with different lengths.

- It is a d-dimensional vector that contains information about a specific position in a sentence. This vector is used to equip each word with information about its position in a sentence. We enhance the model's input to inject the order of words.

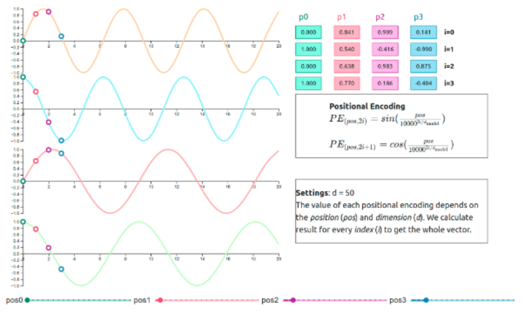
- We use sine and cosine functions of different frequencies:

$$PE_{(pos,2i)} = \sin\left(pos/10000^{2i/d_{model}}\right)$$

$$PE_{(pos,2i+1)} = \cos\left(pos/10000^{2i/d_{model}}\right)$$

Where pos represents the position of tokens and i represents the dimension. Each dimension of the positional encoding corresponds to a different sinusoid wave.

- This is example for sentence with a maximum length of 50 and 128-dimensional positional encoding. Each row represents the embedding vector.



p0	p1	p2	p3	
0.000	0.841	0.909	0.141	i=0
1.000	0.540	-0.416	-0.990	i=1
0.000	0.638	0.983	0.875	i=2
1.000	0.770	0.186	-0.484	i=3

Positional Encoding

$$PE_{(pos,2i)} = sin\left(\frac{pos}{10000^{2i/d_{model}}}\right)$$

$$PE_{(pos,2i+1)} = cos\left(\frac{pos}{10000^{2i/d_{model}}}\right)$$

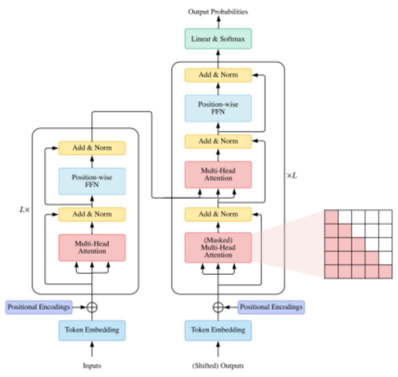
Settings: d = 50

The value of each positional encoding depends on the *position* (*pos*) and *dimension* (*d*). We calculate result for every *index* (*i*) to get the whole vector.

[12]

III. Architecture

- The Transformer is sequence-to-sequence model and consists of an encoder and a decoder, each of which is a stack of L identical blocks.



[12]

b. Encoder module

- Each encoder block is mainly composed of a multi-head self-attention module and a position-wise feed-forward network(FFN). For building a deeper model, a residual connection is employed around each module, followed by Layer Normalization module.

- In the case of text translation, the Encoder module receives an embedding input that is generated based on the input's meaning and position information via the Embedding and Position Encoding layers. From the embedding input, three parameter matrices are created, namely the Query, Key, and value matrices, along with positional information, which are passed through the "Multi-Head Attention" layer.

- Following this step, the Feed-Forward layer addresses the issue of rank collapse that can arise during the computation process. Additionally, a normalization layer is applied to each step, which reduces the dependencies between layers by normalizing the weights used in gradient computation within each layer. To address the issue of vanishing gradients, the Residual Connection is applied to every output of both the attention and feed-forward layers.

c. Decoder module

- Compared to the encoder blocks, decoder blocks additionally insert cross-attention modules between the multi-head self-attention modules and the position-wise FFNs. Furthermore, the self-attention module in the decoder is adapted to prevent each position from attending to subsequent positions.

- The Decoder module in the transformer architecture is similar to the Encoder module, with inclusion of additional layers such as Masked Multi-Head Attention. In addition to the Feed-Forward , Multi-Head Attention, Residual connection, and Add and Norm layers, the Decoder also contains Masked Multi-Head Attention layers.

- The Attention mechanism is applied twice in the decoder : one for computing attention between elements of the targeted output and another for finding attention between the encoding inputs and targeted output. Each attention vector is passed through the feed-forward unit to make the output more comprehensible to the layer. The generated decoding result is then caught by Linear and SoftMax layers at the top of the Decoder to compute the final output of the transformer architecture. This process is repeated multiple times until the last token of a sentence is found.

IV. Strength

a. Self-attention

- Main Advantage of Transformer is use of the attention mechanism to model the global dependencies among nodes within input data. The Attention mechanism captures contextual relationships and enables all elements of an input sequence simultaneously, facilitating efficient parallel computation. Transformers process entire sequences in parallel, significantly boosting training speed and making them highly efficient for large-scale data processing.

b. Effective Management of Long-Term Dependency.

- Traditional RNN-based models struggle with dependencies between distant tokens due to their sequential nature. Unlike these models, Transformers leverage the Self-Attention mechanism to assess the connections between all tokens in an input sequence. This allows them to manage long-Term Dependency effectively.

c. Wide Applicability Across Various NLP Tasks

- Transformers serve as versatile models applicable to a broad range of natural language processing tasks, including text generation, machine translation, summarization, text classification, and question answering. Their adaptable architecture allows them to perform well across different tasks, often state-of-the-art performance due to their ability to capture complex dependencies and relationships in data.

V. Drawback

a. Complexity

- The complexity of self-attention is O$(L^2 \times D)$, where L is sequence length and D is the feature dimension. This quadratic scaling makes the attention module a bottleneck when dealing with long sequences. When generating the next token, we need to re-calculate the attention for the entire sequence, even for tokens that have already been processed. Thus generating tokens for a sequence of length L need roughly L^2 computations which can be costly if the sequence length increases.

b. Lack of Structural prior

- Transformers lack some of the inductive biases inherent to CNNs, such as translation equivariance and locality. Self-attention in Transformers does not assume any structural bias within the inputs. Even the order information is also needed to be learned from training data. Therefore, the absence of inductive biases makes Transformers prone to overfitting, particularly when trained on small or merate-sized datasets.

3. Mamba

I. Preliminary

a. State Space Models

- State Space Models (SSMs) are a traditional mathematical framework used to describe the dynamic behavior of a system over time. SSMs embody the system's behavior through a collection of hidden variables referred to as "states", enabling it to capture temporal data dependencies effectively. In a classical state space model, state equation and observation equation to model the relationships between input $x(t) \in \mathbb{R}$ and output $y(t) \in \mathbb{R}$ at current time t, through a N-dimensional hidden state $h(t) \in \mathbb{R}^N$.

$$h'(t) = Ah(t) + Bx(t),$$
$$y(t) = Ch(t) + Dx(t)$$

Where $h'(t)$ is the derivative of current state $h(t)$, $A \in \mathbb{R}^{N\times N}$ is the state transition matrix that describes how states change over time, $B \in \mathbb{R}^{N\times 1}$ is the input matrix that controls how inputs affect state changes, $C \in \mathbb{R}^{1\times N}$ denotes the output matrix that indicates how outputs are generated based on current states, and $D \in \mathbb{R}$ represents the command coefficient that determines how inputs affect output directly. It can be recognized as a skip connection in deep learning models. In general, most SSMs exclude the second term in observation equation. It means that we set $Dx(t) = 0$.

b. Discretization for creating recurrent and convolution representations.

- To meet the requirements of machine learning settings, SSMs need to undergo a process of discretization, which transforms continuous parameters into discrete parameters. We can apply the Zero-order hold. It assumes that the function value remains constant over the interval $\Delta = [t_{k-1},\ t_k]$, where k is discrete time step.

$$Discretized\ matrix\ A\text{:}\quad \bar{A} = \exp(\Delta \mathrm{A})$$
$$Discretized\ matrix\ B\text{:}\quad \bar{B} = (\Delta \mathrm{A})^{-1}(\exp(\Delta \mathrm{A}) - \mathrm{I}) \cdot \Delta \mathrm{B}$$

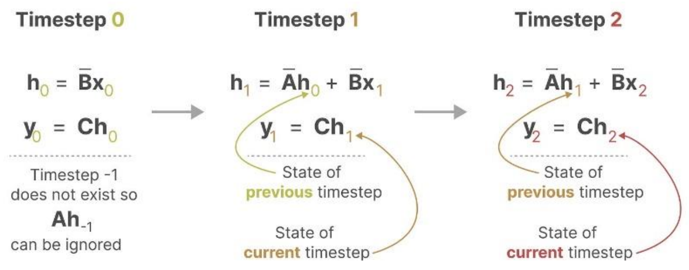
- Discretization allows a transition from continuous SSM to a discrete SSM, representing a shift from a function-to-function formulation $x(t) \rightarrow y(t)$ to a sequence-to-sequence approach $x_k \rightarrow y_k$. The SSM equations ca be rewritten by

$$h_k = \bar{A}h_{k-1} + \bar{B}x_k$$
$$y_k = Ch_k$$

c. Computation

After the parameters have been transformed from $(\Delta, A, B, C) \rightarrow (\bar{A}, \bar{B}, C)$, the model can be computed in two ways. We have efficient inference with the recurrent SSM and parallelizable training with the convolutional SSM. We use the convolutional representation which can be parallelized and during inference, we utilize the efficient recurrent representation.

- The recurrent representation
  - At each timestep, we calculate how the current input influences the previous state and then calculate the predicted output.(Ch_k)



Timestep 0
$h_0 = \bar{B}x_0$
$y_0 = Ch_0$
Timestep -1 does not exist so
Ah_{-1}
can be ignored
Timestep 1
$h_1 = \bar{A}h_0 + \bar{B}x_1$
$y_1 = Ch_1$
State of previous timestep
State of current timestep
Timestep 2
$h_2 = \bar{A}h_1 + \bar{B}x_2$
$y_2 = Ch_2$
State of previous timestep
State of current timestep

[25]

- The Convolutional representation
  - It can calculate the output at each time step independently as follows:

$$y_0 = C\bar{A}^0\bar{B}x_0$$

$$y_1 = C\bar{A}^1\bar{B}x_0 + C\bar{A}^0\bar{B}x_1$$

$$y_2 = C\bar{A}^2\bar{B}x_0 + C\bar{A}^1\bar{B}x_1 + C\bar{A}^0\bar{B}x_2$$

$$\dots$$

$$y_k = C\bar{A}^k\bar{B}x_0 + C\bar{A}^{k-1}\bar{B}x_1 + \cdots + C\bar{A}^1\bar{B}x_{k-1} + C\bar{A}^0\bar{B}x_k$$

  By defining a set of convolutional kernel $\bar{K} = (C\bar{B}, \dots, C\bar{A}^k\bar{B}, \dots)$, the recurrent computation can be reformulated into a convolutional form as: $y = x * \bar{K}$ where x = $[x_0, x_1, \dots]$ and y = $[y_0, y_1, \dots] \in \mathbb{R}^L$ denote the input and the output sequences, while L is the sequence length.

  A key advantage of representing the SSM as a convolution is that it can be trained in parallel like CNNs. However, due to fixed kernel size, their inference is not as fast and unbounded as RNNs.

d. Linear Time Invariance(LTI)

- SSM representations share an important property, namely that of Linear Time Invariance. LTI states the SSMs parameters, A, B and C, are fixed for all timesteps. This means that matrices A, B, and C are the same for every token generated by the SSM, resulting in a static representation that lacks content-awareness.

- LTI models exhibit specific failure modes in tasks like selective copying and induction heads. In the selective copying task, the goal of the SSM is to copy parts of the input and output them in order. In the induction heads task, the goal is to reproduce found in the input.

- From the recurrent view, their constant dynamics transitions cannot let them select the correct information from their context, or affect the hidden state passed along sequence in an input-dependent way. From the convolutional view, it is known that global convolutions have difficulty with the selective copying task because of lack of content-awareness. More concretely, the spacing between inputs-to-outputs is varying and cannot be modeled by static convolution kernels. Matrix A plays a important role as it captures information from previous state to build new state. To enhance this ability, the HiPPO matrix is used to represent matrix A.

II. Mamba – A selective SSM.

a. Selective copying

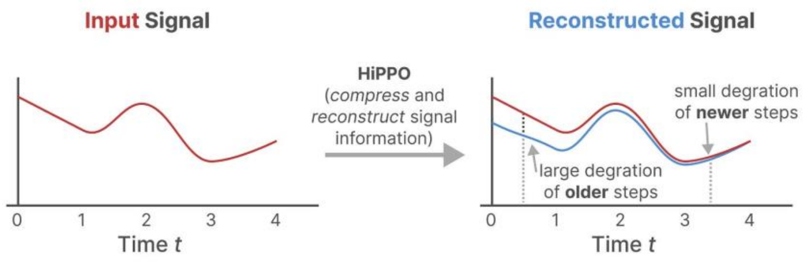
- The selection mechanism is designed to overcome the limitations of LTI models. Conventional state space models are unable to produce personalized outputs based on specific model inputs due to the property of Time Invariance. Mamba designs a time-varying selection mechanism that parameterizes the weight matrices according to model input. Such innovation empowers SSMs to filter out extraneous information while retaining pertinent details indefinitely.

b. HiPPO for Handling Long-Range Dependencies

- The HiPPO matrix is used for the transition matrix A. HiPPO specifies a class of matrices $A \in \mathbb{R}^{N\times N}$ that when incorporated, enable the state $x(t)$ to effectively memorize the history of the input.

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

- Building matrix A using HiPPO was shown to be much better than initializing it as a random matrix. Mathematically, it does so by tracking the coefficients of a Legendre polynomial which allow it to approximate the entire previous history of the sequence.



[25]

c. Selectively Retain Information

- The recurrent representation of an SSM creates a compact state that is quite efficient as it compress the entire history. However, this compression can be less powerful compared to a Transformer model, which retains the full history without compression. Mamba aims to combine the advantages of both approaches: A compact state that is as powerful as the sate of a Transformer. To achieve this selective compression of information, Mamba introduces input-dependent parameters.

- In conventional SSMs, the matrices A, B, and C are independent of the input since there dimensions-N(Hidden state size) and D(size of input vector)-are static. Mamba makes matrices B, C and the step size dependent on the input by incorporating the sequence and batch size of the input. This means that each input token can have different B and C matrices, solving the problem of content-awareness by selectively determining which information to retain in the hidden state.

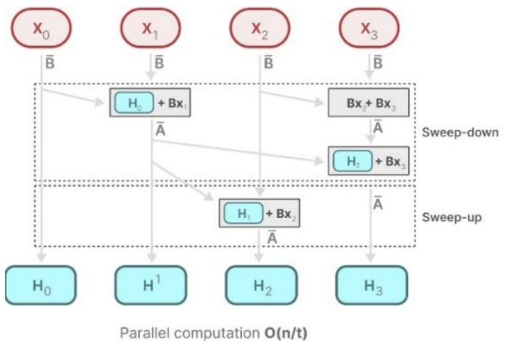
- A smaller step size results in model relying more on the previous context, effectively ignoring specific input words, while a larger step size focuses on the input words more than the context

$$\bar{A} = \exp(\Delta A), \quad \bar{B} = (\Delta A)^{-1}(\exp(\Delta A) - I) \cdot \Delta B .$$

As Δ approaches zero, the current hidden state is preserved. Conversely as Δ approaches infinity, the influence of the current input increases.

d. Scan operation

- System matrices are not input-independent, making it impossible to create a fixed kernel in advance. The convolution representation assumes a fixed kernel. So, they cannot be calculated since these matrices are now dynamic. Mamba makes this possible through the parallel scan algorithm. It assumes the order in which we do operations does not matter through the associate property. As a result, we can calculate the sequences in parts and iteratively combine them:



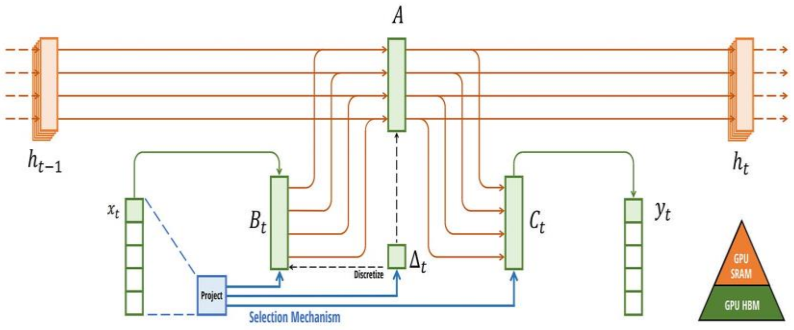
[25]

- Together, dynamic matrices B and C, and the parallel scan algorithm create the selective scan algorithm to represent the dynamic and fast nature of using the recurrent representation. It allows for matrix multiplication using multiple threads, reducing the computational complexity to $O(n/t)$

e. Kernel Fusion to Overcome the computational bottleneck.

- A disadvantage of recent GPUs is their limited transfer speed between their small but highly efficient SRAM and their large but less efficient DRAM, creating a bottleneck when frequently moving data between these memory types. Mamba attempts to limit the number of times we need to go from DRAM to SRAM and vice versa. kernel fusion allows the model to prevent writing intermediate results and continuously performing computations until it is done. Discretization step with step size, Selective scan algorithm and multiplication with C are fused into one kernel. In other words, we transfer parameters with dimensions from HBM to SRAM. Discretization is performed directly within the SRAM. This step ensures efficient handling of data without frequent memory swaps. After processing, the data is transferred back from SRAM to HBM. This approach reduces input/output operations.

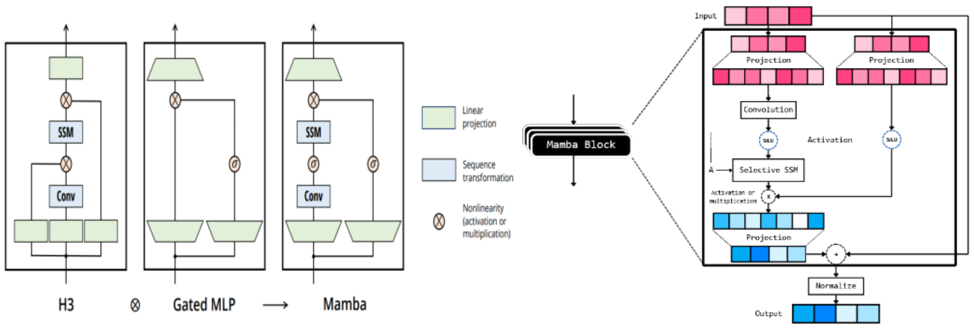
- The intermediate states are not saved but are necessary for the backward pass. Mamba recompute those intermediate states during the backward pass when the input are loaded from HBM to SRAM, which ensures that the fused selective scan layer maintains the same memory efficiency.



[13]

III. architecture

- The Mamba architecture can be broken into a few key components which are connected as:



[17]

- The mamba architecture consists of multiple stacked layers of Mamba blocks. Like the decoder, we can stack multiple blocks and use their output as the input for the next Mamba block. It starts with a linear projection to expand upon the input embeddings. Then, the convolution before the selective SSM plays a crucial role in maintaining contextual coherence.

IV. Strength

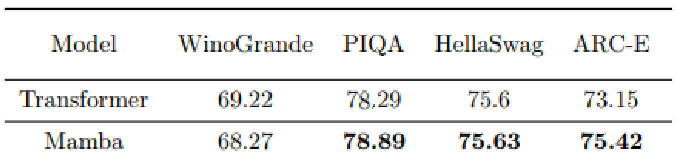
a. SSMs overcome of the shortcoming of Transformers, such as quadratic computational complexity with sequence length and large inference-time memory requirements from the key-value cache. SSMs can capture complex dependencies in sequential data. They can be computed efficiently using either recurrence or convolution operations, achieving linear or near-linear scaling with sequence length, thus reducing the computational costs. One of the variants, Mamba achieves comparable modeling capabilities to Transformers while maintaining linear scalability with sequence length.

b. Conventional Structured State Space Models(SSMs) have shown limited effectiveness in modeling text and other information-dense data impeding their progress in deep learning. Mamba model introduces three innovative techniques based on Structured State Space Models, i.e., High-order Polynomial Projection Operator(HiPPO)-based Memory Initialization, Selection Mechanism, and Hardware-aware Computation.

These techniques aim to enhance the capabilities of SSMs in long-range linear-time sequence modeling. In particular, the initialization strategy establishes a coherent hidden state matrix, effectively facilitating long-range memory. Then the Selection Mechanism empower SSMs to acquire content-aware representations. Lastly Mamba crafts two hardware-aware computation algorithms, Parallel Associative Scan and Memory Re-computation, to enhance training efficiency.

c. Mamba builds on this line of research, inheriting the philosophy of Deep Structured State Space Models(SSMs). Mamba not only achieves high performance but also retains the advantage of fast inference, demonstrating particularly strong results in language modeling. To address the performance issues inherent in SSMs, Mamba introduces selectivity, which enhances performance. The potential computational speed issues arising from this are mitigated through Hardware-aware Parallel Scan, ensuring efficient operations.

d. Mamba model can match or exceed Transformer models on common tasks. Pure SSM models achieve higher accuracy than Transformers when averaged over the WinoGradne, PIQA, HellaSwag, ARC-Easy and ARC-Challenge evaluation.



Model	WinoGrande	PIQA	HellaSwag	ARC-E
Transformer	69.22	78.29	75.6	73.15
Mamba	68.27	**78.89**	**75.63**	**75.42**

[21]

V. Drawback

a. From a computational standpoint, despite the work that went into making Mamba fast, it is still much less hardware-efficient than mechanism such as attention. The missing piece is that modern accelerators such as GPUs and TPUs are highly specialized for matrix multiplications. While this isn't a problem for inference, which is bottlenecked by somewhat different considerations, this can be a big deal during training time.

b. The Mamba architecture still encounters challenges, such as memory loss, generalization to diverse tasks, and inferior capability to capture complex patterns to Transformer-based language models. And Scaling the SSMs to large network sizes is still an open issue, especially with Mamba which has certain stability issues at scale. The stability of SSMs at large network sizes is an open issue. [20, 21]

c. The pure SSM models match or exceed the capabilities of their Transformer counterparts on most downstream tasks but are challenged by tasks that require context-based information retrieval(e.g., copying) and in-context learning. Mamba models underperform compared to Transformer models on tasks requiring strong copying or in-context learning abilities(e.g., five-shot MMLU, Phonebook Lookup). Detailed information can be found in the Drawback section of Mamba-2.

VI. Future directions

a. Current multi-modal algorithms usually directly encode and fuse the multiple cues in a unified Transformer network. Thus, the cost of the inference phase may be twice compared with a single modality only. How to design new SSMs-based backbone for cost-sensitive multi-modal learning is an important research topic.

4. Mamba-2

I. Introduction

a. Transformer, which have played a crucial role in the success of deep learning for various areas. To enable state space model to access and benefit from the valuable techniques initially developed for Transformers. Mamba-2 have introduced a comprehensive framework called SSD, which establishes theorical connection between SSMs and different forms of attention. Formally $y = SSD(A, B, C)(x)$

b. Based on SSD, Mamba-2 has devised a more hardware-efficient computation through a block decomposition matrix multiplication algorithm. By viewing state space models as semi-separable matrices through the matrix transformation, Mamba-2 divides the computation into matrix blocks. Diagonal blocks represent intra-chunk computations, while the off-diagonal blocks represent inter-chunk computations factored through the SSM's hidden state.

c. Mamba-2 allows for Sequence Parallelism by passing the recurrent state between multiple GPUs, and Tensor Parallelism is possible because of independent parallel projections of A, B, C, and X inputs of its SSM part.

The authors Albert Gu and Tri Dao aim to solve the conceptual connections between state space models and attention. Additionally, They explored whether Mamba models' training could be accelerated through matrix multiplication techniques.

II. Component

a. Structured State Space Duality(SSD)

- The SSD is based on SSMs. The equations for the SSM are given as:

$$h_k = Ah_{k-1} + Bx_k$$
$$y_k = Ch_k$$

- In the selective SSM, the matrix A_t is constrained to $a_t \times I$ (a scalar multiplied by the identity matrix). This approach can transform the temporal information of matrix A_t into the mask matrix L, which represents the state transition information form step j to i.

$$L_{ij}{=}a_i * a_{i-1} * \dots * a_{j+1},$$

1-SS matrices lies in their equivalence to the minimal form of a scalar recurrence.

$$y_t = a_{t:0}x_0 + \cdots + a_{t:t}x_t = a_t(a_{t-1:0}x_0 + \cdots + a_{t-1:t-1}x_{t-1}) + a_{t:t}x_t = a_t y_{t-1} + x_t$$

b. SSM with Matrix Transformations

- Since GPUs/TPUs are optimized for matrix operations, the SSM can be transformed into matrix forms similar to the attention mechanism:

$$Y = M * X$$

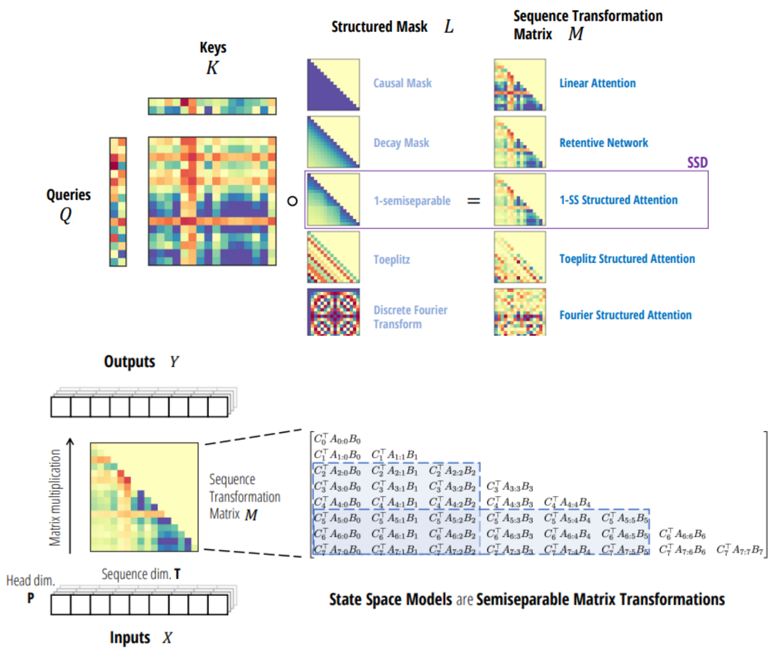
$$M = L \circ (C * B^T)$$

In this transformation, C can be seen as the Query, B as the Key, L as the Mask Matrix.

- The dual form of SSD is a quadratic computation closely related to attention, defined as

$$(L \circ QK^T) \cdot V, \qquad L_{ij} = \begin{cases} a_i \times \cdots \times a_{j+1}, & i \geq j \\ 0, & i < j \end{cases}$$

where a_i are input-dependent scalars bounded within [0, 1]. Compared to standard attentions, there are two main differences. The softmax operation is dropped. The attention matrix is element-wise multiplied by an additional mask matrix L.


Keys K
Queries Q
Structured Mask L
Sequence Transformation Matrix M
Causal Mask
Linear Attention
Decay Mask
Retentive Network
SSD
1-semiseparable
1-SS Structured Attention
Toeplitz
Toeplitz Structured Attention
Discrete Fourier Transform
Fourier Structured Attention
Outputs Y
Matrix multiplication
Sequence Transformation Matrix M
Head dim. P
Sequence dim. T
Inputs X
State Space Models are Semiseparable Matrix Transformations

[16]

- Therefore, after completing the operations with L in SSD, the matrix M takes the form shown in the diagrams above. This transformation allows us to understand why the SSM can be converted to a form similar to the attention mechanism, which captures correlations between tokens, thereby achieving good performance. Additionally, Employing Matrix Transformations, enables the development of algorithms that are more computationally efficient.

c. SSD algorithm

- To achieve parallel processing and memory efficiency, computations can be divided into chunks. As seen in the figure below, the blue regions below the diagonal represent submatrices that are low-rank. In contrast, the main diagonal corresponds to the direct relationship between the state and input at a specific time point.

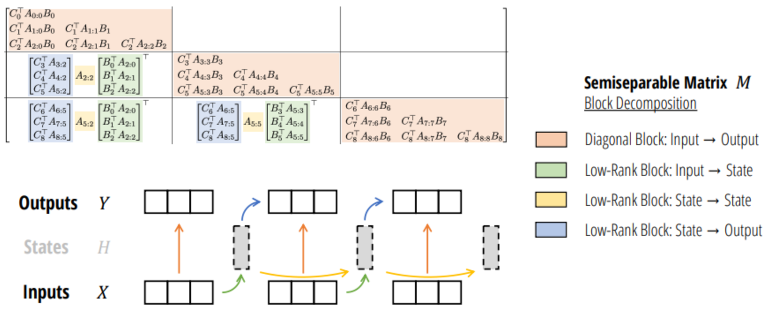
$$
M = \left[\begin{array}{ccc|ccc|ccc}
C_0^\top A_{0:0}B_0 & & & & & & & & \\
C_1^\top A_{1:0}B_0 & C_1^\top A_{1:1}B_1 & & & & & & & \\
C_2^\top A_{2:0}B_0 & C_2^\top A_{2:1}B_1 & C_2^\top A_{2:2}B_2 & & & & & & \\
\hline
C_3^\top A_{3:0}B_0 & C_3^\top A_{3:1}B_1 & C_3^\top A_{3:2}B_2 & C_3^\top A_{3:3}B_3 & & & & & \\
C_4^\top A_{4:0}B_0 & C_4^\top A_{4:1}B_1 & C_4^\top A_{4:2}B_2 & C_4^\top A_{4:3}B_3 & C_4^\top A_{4:4}B_4 & & & & \\
C_5^\top A_{5:0}B_0 & C_5^\top A_{5:1}B_1 & C_5^\top A_{5:2}B_2 & C_5^\top A_{5:3}B_3 & C_5^\top A_{5:4}B_4 & C_5^\top A_{5:5}B_5 & & & \\
\hline
C_6^\top A_{6:0}B_0 & C_6^\top A_{6:1}B_1 & C_6^\top A_{6:2}B_2 & C_6^\top A_{6:3}B_3 & C_6^\top A_{6:4}B_4 & C_6^\top A_{6:5}B_5 & C_6^\top A_{6:6}B_6 & & \\
C_7^\top A_{7:0}B_0 & C_7^\top A_{7:1}B_1 & C_7^\top A_{7:2}B_2 & C_7^\top A_{7:3}B_3 & C_7^\top A_{7:4}B_4 & C_7^\top A_{7:5}B_5 & C_7^\top A_{7:6}B_6 & C_7^\top A_{7:7}B_7 & \\
C_8^\top A_{8:0}B_0 & C_8^\top A_{8:1}B_1 & C_8^\top A_{8:2}B_2 & C_8^\top A_{8:3}B_3 & C_8^\top A_{8:4}B_4 & C_8^\top A_{8:5}B_5 & C_8^\top A_{8:6}B_6 & C_8^\top A_{8:7}B_7 & C_8^\top A_{8:8}B_8
\end{array}\right]
$$

$$
= \left[\begin{array}{c|c|c}
\begin{matrix} C_0^\top A_{0:0}B_0 & & \\ C_1^\top A_{1:0}B_0 & C_1^\top A_{1:1}B_1 & \\ C_2^\top A_{2:0}B_0 & C_2^\top A_{2:1}B_1 & C_2^\top A_{2:2}B_2 \end{matrix} & & \\
\hline
\begin{bmatrix} C_3^\top A_{3:2} \\ C_4^\top A_{4:2} \\ C_5^\top A_{5:2} \end{bmatrix} A_{2:2} \begin{bmatrix} B_0^\top A_{2:0} \\ B_1^\top A_{2:1} \\ B_2^\top A_{2:2} \end{bmatrix}^\top & \begin{matrix} C_3^\top A_{3:3}B_3 & & \\ C_4^\top A_{4:3}B_3 & C_4^\top A_{4:4}B_4 & \\ C_5^\top A_{5:3}B_3 & C_5^\top A_{5:4}B_4 & C_5^\top A_{5:5}B_5 \end{matrix} & \\
\hline
\begin{bmatrix} C_6^\top A_{6:5} \\ C_7^\top A_{7:5} \\ C_8^\top A_{8:5} \end{bmatrix} A_{5:2} \begin{bmatrix} B_0^\top A_{2:0} \\ B_1^\top A_{2:1} \\ B_2^\top A_{2:2} \end{bmatrix}^\top & \begin{bmatrix} C_6^\top A_{6:5} \\ C_7^\top A_{7:5} \\ C_8^\top A_{8:5} \end{bmatrix} A_{5:5} \begin{bmatrix} B_3^\top A_{5:3} \\ B_4^\top A_{5:4} \\ B_5^\top A_{5:5} \end{bmatrix}^\top & \begin{matrix} C_6^\top A_{6:6}B_6 & & \\ C_7^\top A_{7:6}B_6 & C_7^\top A_{7:7}B_7 & \\ C_8^\top A_{8:6}B_6 & C_8^\top A_{8:7}B_7 & C_8^\top A_{8:8}B_8 \end{matrix}
\end{array}\right]
$$

•

[16]

Consequently, as depicted in the figure, computations can be performed accordingly. All operations, except for those involving the yellow blocks (which map the state to the next state), can be parallelized, allowing for efficient computation.

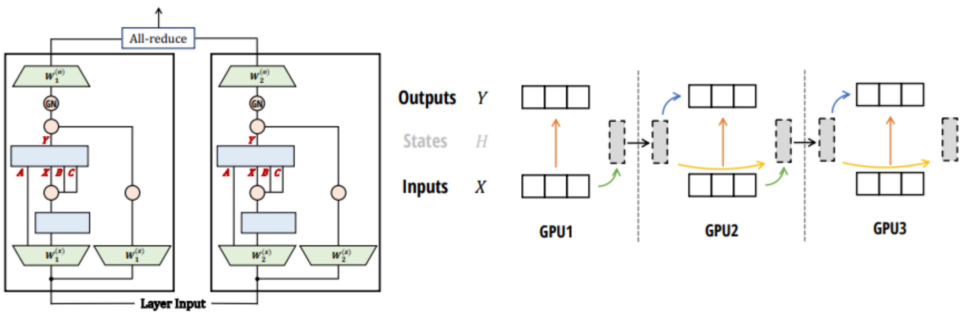


Semiseparable Matrix M
Block Decomposition
Diagonal Block: Input → Output
Low-Rank Block: Input → State
Low-Rank Block: State → State
Low-Rank Block: State → Output
Outputs Y
States H
Inputs X

[16]

d. Parallelism with the Mamba-2 Block

- By leveraging the characteristics of the SSD algorithm and the Mamba-2 block, two types of distributed processing can be achieved. Distributing model parameters across multiples GPUs to accelerate the training process. Distributing input sequences across multiple GPUs for processing.



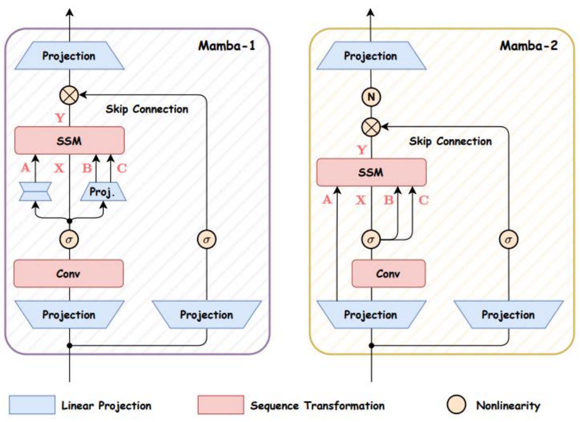
[16]

Unlike the original Mamba, the new approach efficiently utilizes GPUs, enabling effective large-scale model training and processing of long sequence data.

III. Mamba-2 Architecture

a. In the original Mamba block, A, B and C were sequentially computed based on X. However, in Mamba-2, matrix transformation enables the parallel computation of A, B, C and, X directly computed in parallel from the input

b. Mamba-2 simplifies the mamba-1 block by removing sequential linear projections, allowing faster computation of the SSD structure compared to the parallel selective scanning in Mamba-1. A normalization layer is added after the skip connection to improve training stability.



[20]

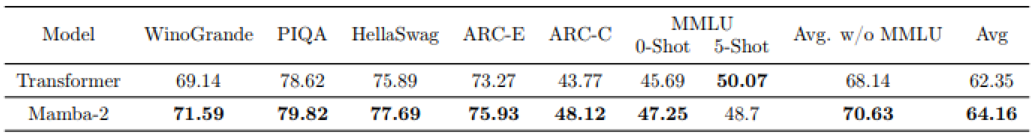
This figure illustrate tensor parallel.

IV. Strength

a. Mamba-2 improves hardware efficiency in computations by employing a block decomposition matrix multiplication algorithm, building upon the SSD framework. By transforming state space models into semi-separable matrices, Mamba-2 breaks down the computation into matrix blocks, where diagonal blocks handle intra-chunk computations and off-diagonal blocks manage inter-chunk computations through the SSM's hidden state. This method allows Mamba-2 to achieve a training speed that is 2-8x faster than Mamba-1's parallel associative scan, while maintaining competitive performance with Transformers.

b. Transformers have benefited from 7 years of systems optimization from the whole research community and large companies. The SSD framework draws connections between SSMs and attention, and allows us to implement many of these optimizations for model like Mamba-2 as well. We focus on tensor parallel and sequence parallel for large-scale training, as well as variable-length sequences for efficient finetuning and inference. In other words, connecting SSMs and attention allows to develop a shared vocabulary and library of technics for Transformer and Mamba. In light of this, an important future direction arises, i.e., to explore how the emerging techniques designed for Transformer-based models can be effectively applied to Mamba-based models.

c. Below table is evaluation results for 8B-parameter models trained on 3.5T tokens. Training for 3.5T tokens allows Mamba-2 to approach Transformer-level accuracy on MMLU and produces a pure SSM model that exceeds the Transformer in terms of average task accuracy.



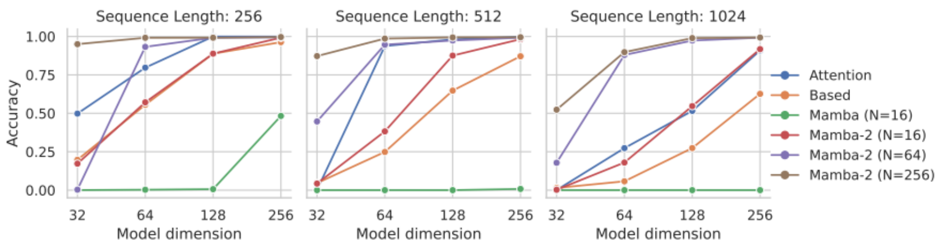
Model	WinoGrande	PIQA	HellaSwag	ARC-E	ARC-C	MMLU 0-Shot	MMLU 5-Shot	Avg. w/o MMLU	Avg
Transformer	69.14	78.62	75.89	73.27	43.77	45.69	**50.07**	68.14	62.35
Mamba-2	**71.59**	**79.82**	**77.69**	**75.93**	**48.12**	**47.25**	48.7	**70.63**	**64.16**

[21]

d. Several experiments conducted in the original paper show that Mamba-2 model perform well compared to Transformer, demonstrating strong performance across various tasks:

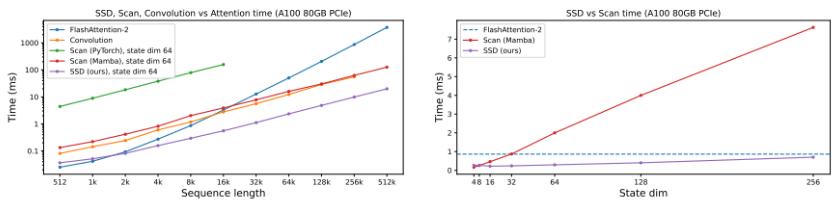
- Multi-Query Associative Recall

  The task is that the model stores multiple key-value pairs and outputs the correct value for a given key. Mamba-2 excel in associative recall tasks, demonstrating better performance than attention mechanisms when handling multiple key-value pairs, especially at higher dimensions. Associative recall tasks are challenging for SSMs, which must memorize all relevant information their recurrent state.



[16]

e. Efficiency Benchmarks



[16]

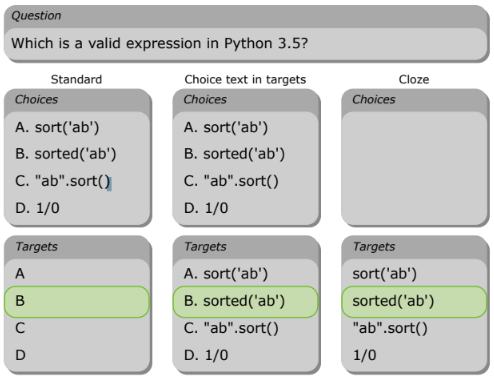
- The SSD algorithm enables faster computations compared to the traditional Scan algorithm. While computational efficiency typically declines as the state dimension increase, SSD improves processing by converting these operations into matrix computations, leading to significantly enhanced efficiency.

V. Drawback

b. The pure SSM models match or exceed the capabilities of their Transformer counterparts on most downstream tasks but are challenged by tasks that require context-based information retrieval(e.g., copying) and in-context learning

Mamba models underperform compared to Transformer models on tasks requiring strong copying or in-context learning abilities(e.g., five-shot MMLU, Phonebook Lookup). Although Mamba-2 models are good at modeling language, they struggle with in-context learning and recalling information form the context compared to Transformer models. Pure SSM models are unable to directly route the knowledge of each answer into a single answer token. In contrast, the self-attention layers in the Transformer are particularly good at that task, especially when they are shown several in-context examples that teach them to do such routing

The below figure is illustration of the MMLU. In all cases the model is first prompted with a question. In the standard and 'choice text in targets' variants, the prompt includes four multiple choice answers following the question. The correct answer is then calculated by measuring which of four answers, represented in the target format, is predicted to have the highest probability of following the given prompt.



Question
Which is a valid expression in Python 3.5?

Standard

Choices
A. sort('ab')
B. sorted('ab')
C. "ab".sort()
D. 1/0

Targets
A
B
C
D

Choice text in targets

Choices
A. sort('ab')
B. sorted('ab')
C. "ab".sort()
D. 1/0

Targets
A. sort('ab')
B. sorted('ab')
C. "ab".sort()
D. 1/0

Cloze

Choices

Targets
sort('ab')
sorted('ab')
"ab".sort()
1/0

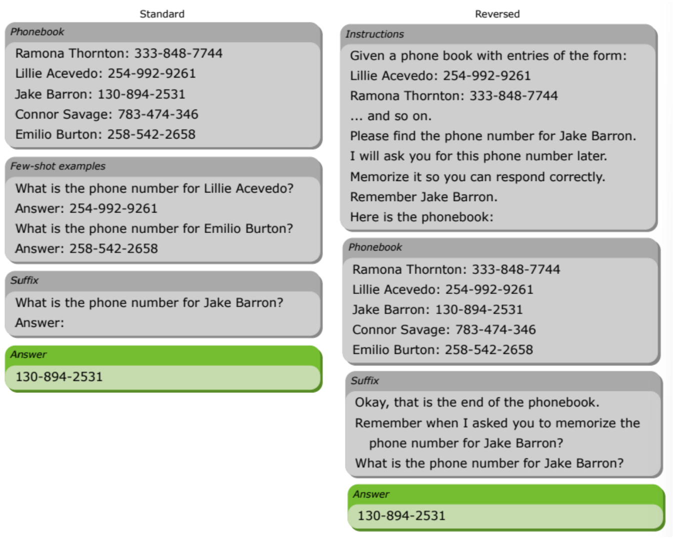
[21]

Results of evaluating our 8B-parameter pure SSM and Transformer models trained on 1.1T tokens on the three formulations of MMLU described below table. While pure SSM models struggle with the standard and choice-text-in-targets formulations on this token horizon.

Model	MMLU-Standard 0-Shot	MMLU-Standard 5-Shot	MMLU-W/Choice 0-Shot	MMLU-W/Choice 5-Shot	MMLU-Cloze 0-Shot	MMLU-Cloze 5-Shot
Transformer	**38.32**	**46.28**	**33.54**	**46.64**	37.26	39.24
Mamba	28.63	28.00	27.42	29.17	**38.26**	**39.28**
Mamba-2	28.94	29.19	28.54	30.68	37.68	38.17

[21]

- In evaluations of pure SSM-based model comparison to Transformers on the synthetic Phonebook tasks, it aims to measure a model's ability to perform in-context learning and copying from earlier in the context. In this task, the model is first prompted with a list of (name, phone number) pairs, and then asked 'What is the phone number for name?' with two example question answer pairs before the actual question used for testing. Names and phone numbers are randomly generated. For each trial, they randomly generate names phone numbers to create the phone book and randomly select which names are used for and the two examples and the final query. Accuracy on tasks is then measured by whether the model generates the correct phone numbers or not.



Standard

Phonebook
Ramona Thornton: 333-848-7744
Lillie Acevedo: 254-992-9261
Jake Barron: 130-894-2531
Connor Savage: 783-474-346
Emilio Burton: 258-542-2658

Few-shot examples
What is the phone number for Lillie Acevedo?
Answer: 254-992-9261
What is the phone number for Emilio Burton?
Answer: 258-542-2658

Suffix
What is the phone number for Jake Barron?
Answer:

Answer
130-894-2531

Reversed

Instructions
Given a phone book with entries of the form:
Lillie Acevedo: 254-992-9261
Ramona Thornton: 333-848-7744
... and so on.
Please find the phone number for Jake Barron.
I will ask you for this phone number later.
Memorize it so you can respond correctly.
Remember Jake Barron.
Here is the phonebook:

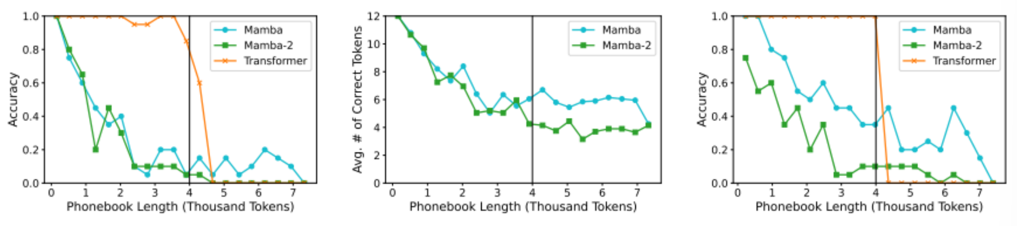
Phonebook
Ramona Thornton: 333-848-7744
Lillie Acevedo: 254-992-9261
Jake Barron: 130-894-2531
Connor Savage: 783-474-346
Emilio Burton: 258-542-2658

Suffix
Okay, that is the end of the phonebook.
Remember when I asked you to memorize the phone number for Jake Barron?
What is the phone number for Jake Barron?

Answer
130-894-2531

[21]

- In this situation, the synthetic phonebook task has two versions. In the standard configuration, the model is first prompted with a phone book and then shown two in-context example questions before the final question which asks it to recall the phone book number for a specific person. In a reversed setting, the model is first told which phone number to look for before being shown the phone book and is later asked to recall this number.



[21]

- In the first experiment with the standard phonebook task, Transformers are capable of in-context learning and answering questions that require copying from the input, while SSM models struggle with this task. In the second experiment with the standard phonebook task, SSM models exhibit fuzzy memory – while they are unable to correctly predict the phone number, they predict phone numbers that share multiple digits with the correct answer. In the reversed phonebook setting, even when informed of the target number at the beginning, SSM models still perform poorly compared to Transformers, demonstrating continued difficulty in accurate recall.

V. Future Directions

With SSD, we have connected attentionand SSMs making design faster algorithms and implement systems optimizations for SSMs. There are still exiting directions.

a. Understanding[15]

- Hybrid models with a few (4-6) attention layers perform very well, even better than pure Mamba(-2) or Transformer++. What are these attention layers doing? Can they can be replaced with another mechanism?

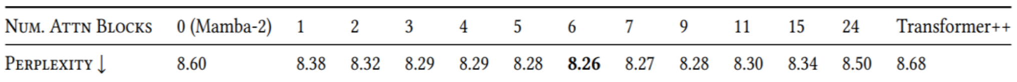
b. Training optimizations[15]

- Though SSD might be faster than attention, Mamba-2 as a whole might still be slower than Transformers at short seqeunce length, since the MLP layers in Transformers are very hardware-friendly. Implement of SSD does not specifically take advantage of new featrues on H100 GPUs, and we look forward to future optimizations that could make SSMs faster to train than Transformers for large-scale pretraining at 2-4K sequence legnth.

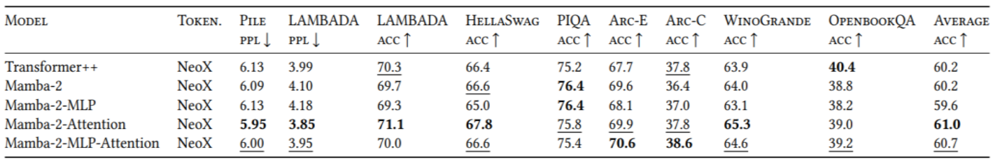
c. Inference optimizations[15]

- There’s a whole suite of optimizations tailored to Transformers, in particular handling the KV cache (quantization, speculative decoding). How would the inference landscape change if model states(e.g. SSM states) no longer scale with context length, and KV cache is no longer the bottleneck?

d. Hybrid Models with SSD, MLP, and Attention



Num. Attn Blocks	0 (Mamba-2)	1	2	3	4	5	6	7	9	11	15	24	Transformer++
Perplexity ↓	8.60	8.38	8.32	8.29	8.29	8.28	**8.26**	8.27	8.28	8.30	8.34	8.50	8.68



Model	Token.	Pile ppl ↓	LAMBADA ppl ↓	LAMBADA acc ↑	HellaSwag acc ↑	PIQA acc ↑	Arc-E acc ↑	Arc-C acc ↑	WinoGrande acc ↑	OpenbookQA acc ↑	Average acc ↑
Transformer++	NeoX	6.13	3.99	70.3	66.4	75.2	67.7	37.8	63.9	**40.4**	60.2
Mamba-2	NeoX	6.09	4.10	69.7	66.6	**76.4**	69.6	36.4	64.0	38.8	60.2
Mamba-2-MLP	NeoX	6.13	4.18	69.3	65.0	**76.4**	68.1	37.0	63.1	38.2	59.6
Mamba-2-Attention	NeoX	**5.95**	**3.85**	**71.1**	**67.8**	75.8	69.9	37.8	**65.3**	39.0	**61.0**
Mamba-2-MLP-Attention	NeoX	6.00	3.95	70.0	66.6	75.4	**70.6**	**38.6**	64.6	39.2	60.7

[21]

- Combining approximately 10% of Attention layers with the SSD layer can achieve even better performance. While adding an MLP layer may slightly reduce model quality, it demonstrates the potential for easier expansion into MoE(Mixture of Experts) models.

References

1. Krizhevsky, A., I. Sutskever, and G.E. Hinton, *Imagenet classification with deep convolutional neural networks.* Advances in neural information processing systems, 2012. **25**.
2. Goodfellow, I.J., J. Shlens, and C. Szegedy, *Explaining and harnessing adversarial examples.* arXiv preprint arXiv:1412.6572, 2014.
3. Goodfellow, I., *Deep learning*. Vol. 196. 2016: MIT press.
4. Vaswani, A., *Attention is all you need.* arXiv preprint arXiv:1706.03762, 2017.
5. Alammar, J., *The Illustrated Transformer*. 2018.
6. Gu, J., et al., *Recent advances in convolutional neural networks.* Pattern recognition, 2018. **77**: p. 354-377.
7. Redmon, J. *The Ancient Secrets of Computer Vision*. 2018.
8. Sharma, N., V. Jain, and A. Mishra, *An analysis of convolutional neural networks for image classification.* Procedia computer science, 2018. **132**: p. 377-384.
9. Voelker, A., Ivana Kajić, and Chris Eliasmith, *Legendre memory units: Continuous-time representation in recurrent neural networks*, in *Advances in neural information processing systems 32*. 2019.
10. Fan, A., et al., *Addressing some limitations of transformers with feedback memory.* arXiv preprint arXiv:2002.09402, 2020.
11. Alzubaidi, L., et al., *Review of deep learning: concepts, CNN architectures, challenges, applications, future directions.* Journal of big Data, 2021. **8**: p. 1-74.
12. Erdem, K.. "*Understanding Positional Encoding in Transformers.*", 2021
13. Lin, T., et al., *A survey of transformers.* AI open, 2022. **3**: p. 111-132.
14. Gu, A. and T. Dao, *Mamba: Linear-time sequence modeling with selective state spaces.* arXiv preprint arXiv:2312.00752, 2023.
15. Islam, S., et al., *A comprehensive survey on applications of transformers for deep learning tasks.* Expert Systems with Applications, 2023: p. 122666.
16. Dao, T., *State Space Duality (Mamba-2)*. 2024.
17. Dao, T. and A. Gu, *Transformers are SSMs: Generalized models and efficient algorithms through structured state space duality.* arXiv preprint arXiv:2405.21060, 2024.
18. Jumle, V., *Mamba: SSM, Theory, and Implementation in Keras and TensorFlow*. 2024.
19. Liu, F. and Q. Li, *From generalization analysis to optimization designs for state space models.* arXiv preprint arXiv:2405.02670, 2024.
20. Patro, B.N. and V.S. Agneeswaran, *Mamba-360: Survey of state space models as transformer alternative for long sequence modelling: Methods, applications, and challenges.* arXiv preprint arXiv:2404.16112, 2024.
21. Qu, H., et al., *A Survey of Mamba.* arXiv preprint arXiv:2408.01129, 2024.
22. Waleffe, R., et al., *An Empirical Study of Mamba-based Language Models.* arXiv preprint arXiv:2406.07887, 2024.

23. Wang, X., et al., *State space model for new-generation network alternative to transformers: A survey.* arXiv preprint arXiv:2404.09516, 2024.
24. Zhao, X., et al., *A review of convolutional neural networks in computer vision.* Artificial Intelligence Review, 2024. **57**(4): p. 99.
25. Grootendorst, M. *"A Visual Guide to Mamba and State Space Models."*, 2024