Transformers to Mamba

Part 1 of 2

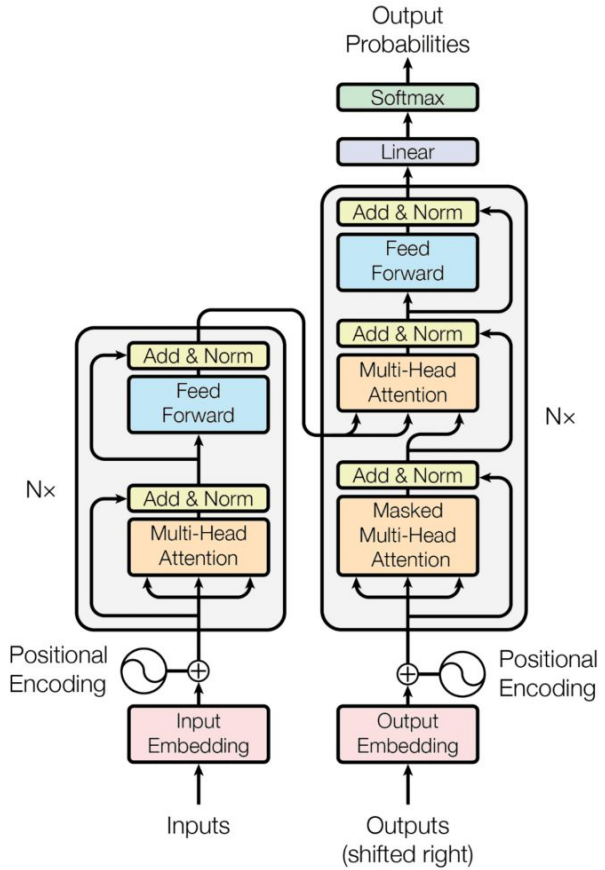
- Basics of transformers
- what's Encoder vs Decoder stack
- Disadvantages (dejavu Paper)
- SSMs basic
- S4 and S5
- Get into mamba

A quick recap of Transformers

the transformer architecture

> proposed : 2017 in “Attention is all you need” by Vaswani et al.

> based on the idea of attention, specifically multi head attention



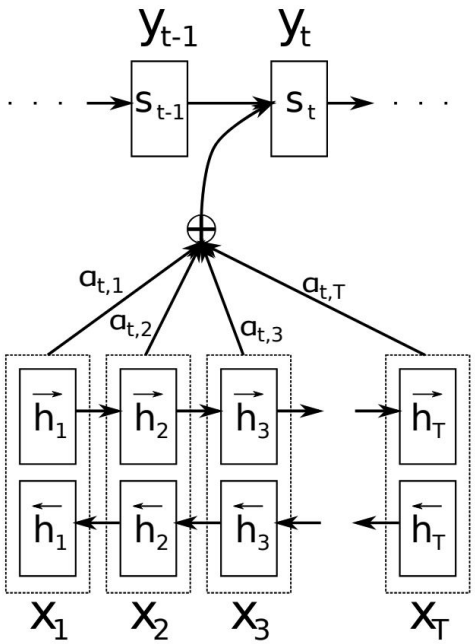
The transformer architecture

origins of attention

> first introduced attention mechanism : Bahdanau / additive attention mechanism for neural machine translation.

> link to paper https://arxiv.org/pdf/1409.0473

do check it out it’s a nice paper!

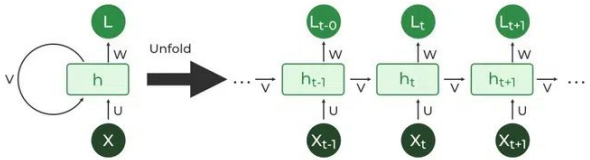


language / sequence modelling before transformers ..

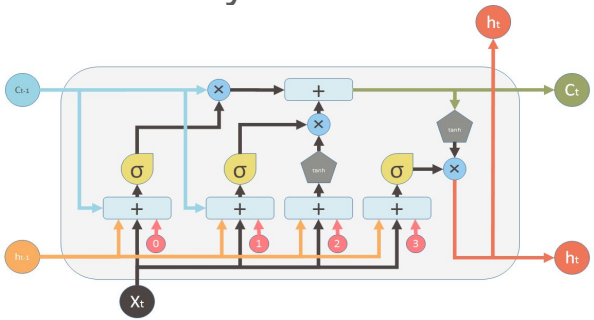
> the RNNs, LSTM, GRUs were used for sequence modeling tasks before the idea of the transformer and were SOTA approaches

>but were somewhat ineffective at LRD tasks

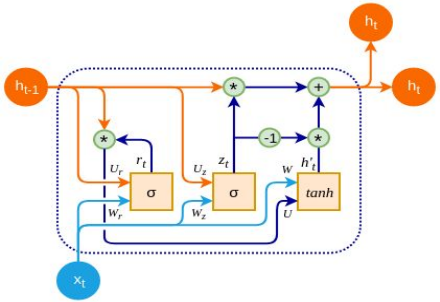
>as they maintain a lot of hidden states, parallelization was very difficult and thus long range sequences were not easy to model



RNN



LSTM

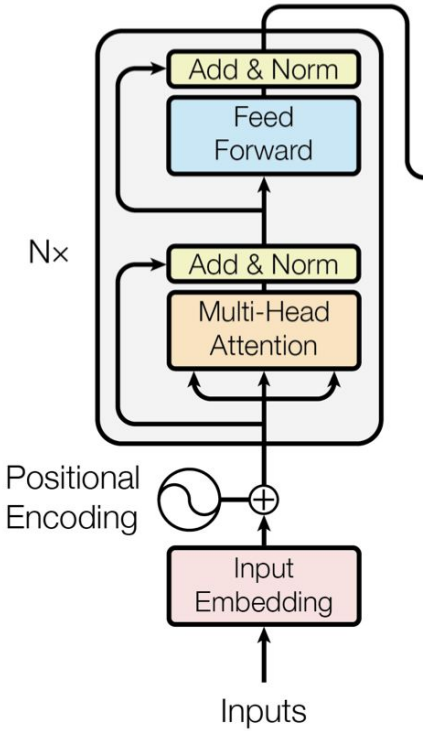


GRU

the encoder stack

> the transformer architecture consists of encoders and decoders which are stacked on top of each other.

> the encoder stack in particular consists of 6 identical layers
> and each layer consists of 2 sub-layers :

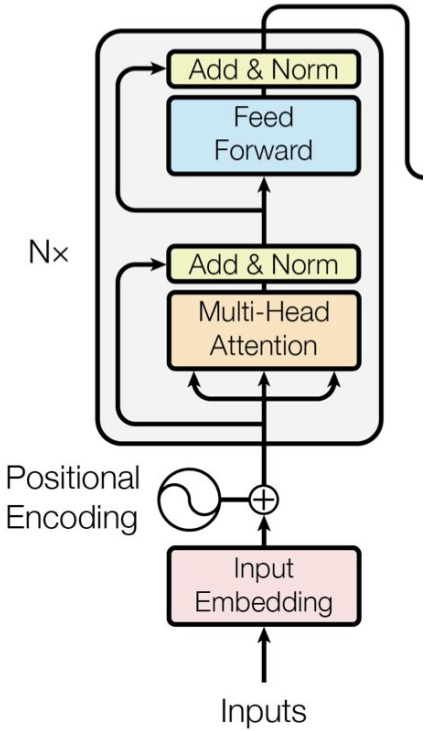


the encoder stack

> the transformer architecture consists of encoders and decoders which are stacked on top of each other.

> the encoder stack in particular consists of 6 identical layers
> and each layer consists of 2 sub-layers :

> 1st is a MHA mechanism

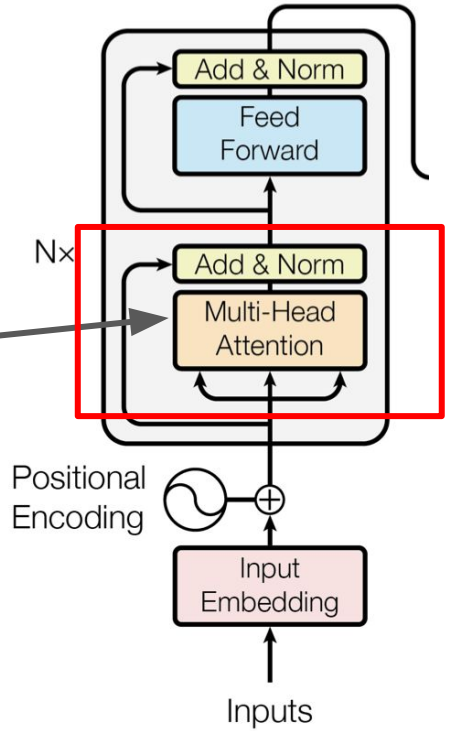


the encoder stack

> the transformer architecture consists of encoders and decoders which are stacked on top of each other.

> the encoder stack in particular consists of 6 identical layers
> and each layer consists of 2 sub-layers :

> 1st is a MHA mechanism (we will get to this soon)



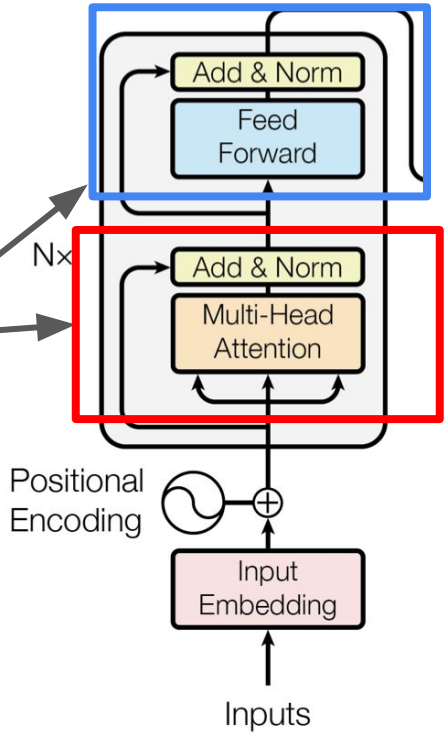
the encoder stack

> the transformer architecture consists of encoders and decoders which are stacked on top of each other.

> the encoder stack in particular consists of 6 identical layers
> and each layer consists of 2 sub-layers :

> 1st is a MHA mechanism

> 2nd being a pointwise feed forward network



the encoder stack

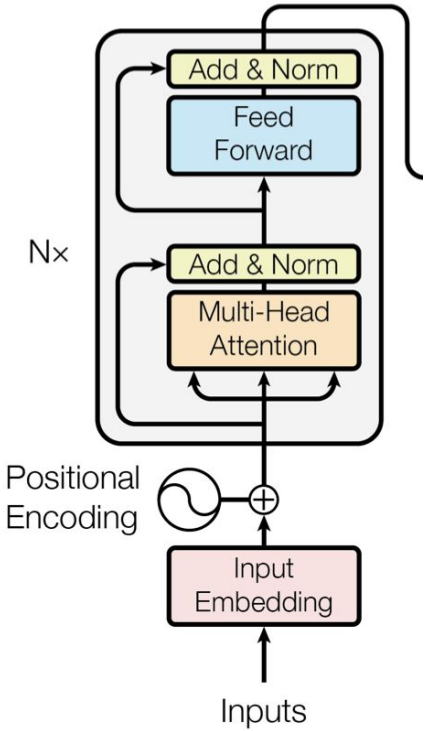
> there is a residual connection around these sub layers

> and which is then further followed by a layer normalization

> and thus the final output of each of the layers then is :

$$\text{LayerNorm}(x + \text{Sublayer}(x))$$

> attention ! : the encoder layers sure do look pretty similar, but they do not share weights!

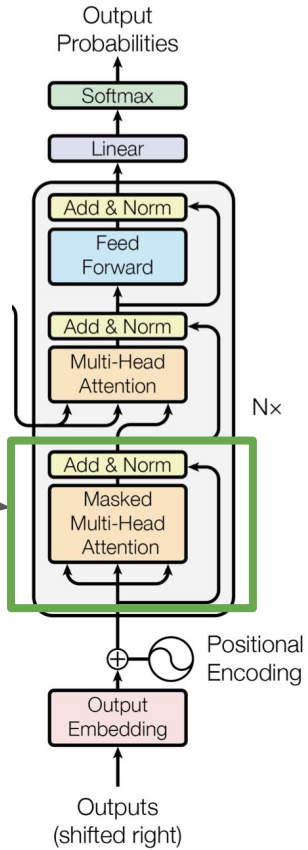


the decoder stack

> the decoder stack, similar to the encoder stack is composed of 6 identical layers containing, this time, 3 sub layers.

> the 3 sub-layers are :

> 1. masked MHA



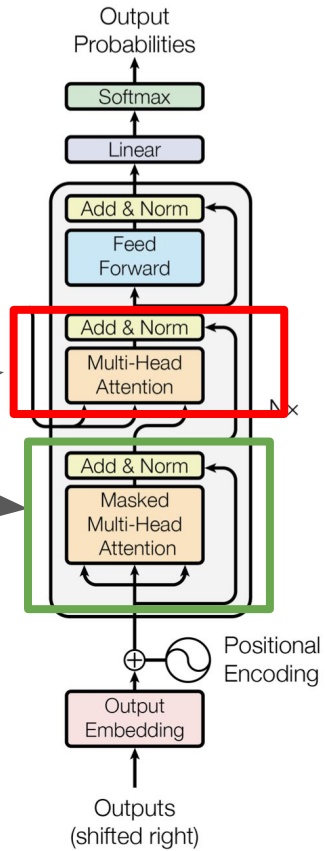
the decoder stack

> the decoder stack, similar to the encoder stack is composed of 6 identical layers containing, this time, 3 sub layers.

> the 3 sub-layers are :

> 1. masked MHA

> 2. MHA layer



the decoder stack

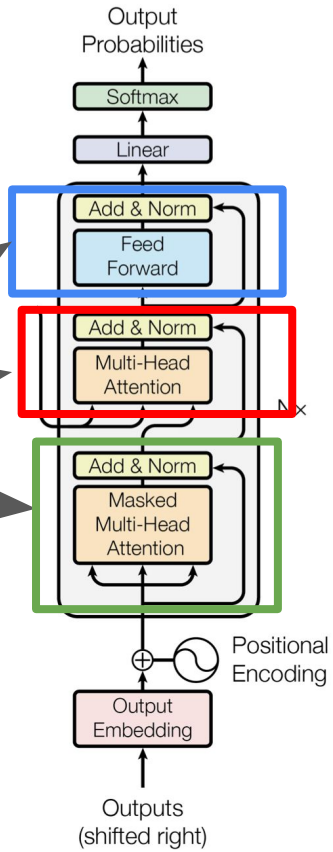
> the decoder stack, similar to the encoder stack is composed of 6 identical layers containing, this time, 3 sub layers.

> the 3 sub-layers are :

> 1. masked MHA

> 2. MHA layer

> 3. pointwise feed forward layer



BASIC SELF-ATTENTION

→ Input: Sq of tensors

↳ $x_1, \ldots x_t$

→ output:

$y_1, y_2, \ldots, y_t$

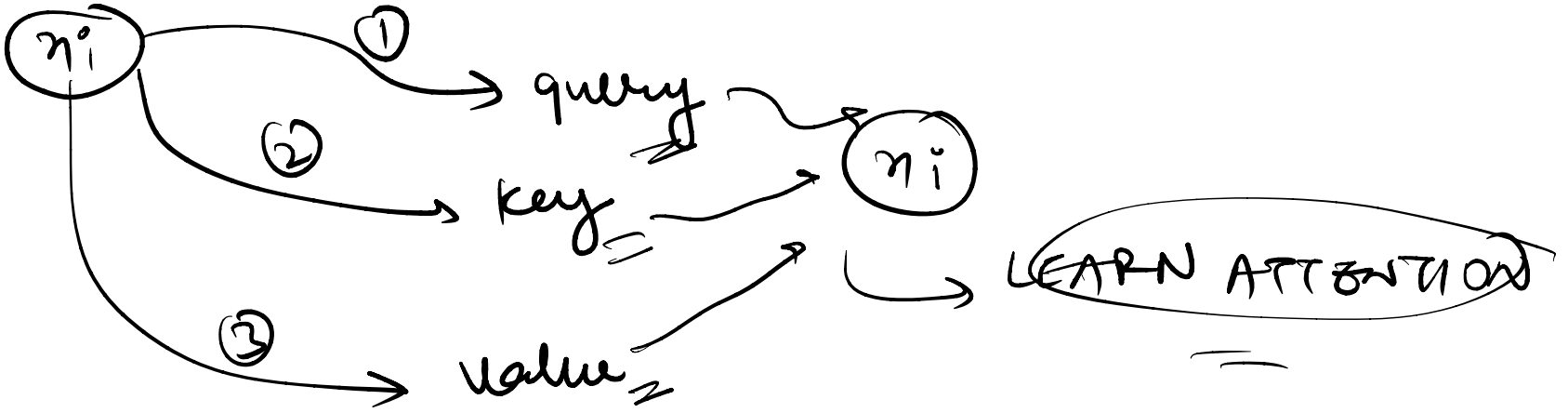
$y_i = \sum_j w_{ij} x_j$

↳ learned

↳ $f(x_i, x_j)$

↳ $w_{ij}' = x_i^T x_j$

⇒ must sum to 1 → $w_{ij} = \frac{\exp(w_{ij}')}{\sum_j \exp(w_{ij}')}$



query: compared to every other vector
↳ attn weights
↳ y_i

key: compared to every other vector
to compute $w_{ij} \rightarrow y_i$

value: summed with other vector to form
the result of the attn weighted sum

$q_i = W_q x_i$, $k_i = W_k x_i$, $v_i = W_v x_i$

$$\Rightarrow \quad w_{ij}' = q_i^T k_j$$

$$\Rightarrow \quad w_{ij} = \text{softmax}(w_{ij}')$$

$$y_i = \sum_j w_{ij} v_j$$

the masked MHA layer in the decoder stack

> the masked MHA in the decoder is one of the most crucial aspects of the decoder which makes the self-attention work correctly.

> by masking we ensure that each position in the sequence can only attend to the positions before it.

> which is highly crucial in language modelling and seq-to-seq tasks such as machine translation

the pointwise feed forward networks

> the pointwise feed forward networks consists of two linear transformations with a ReLU activation in between and are present in each of the layers in the encoder and the decoder

$$\mathrm{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2 \tag{2}$$

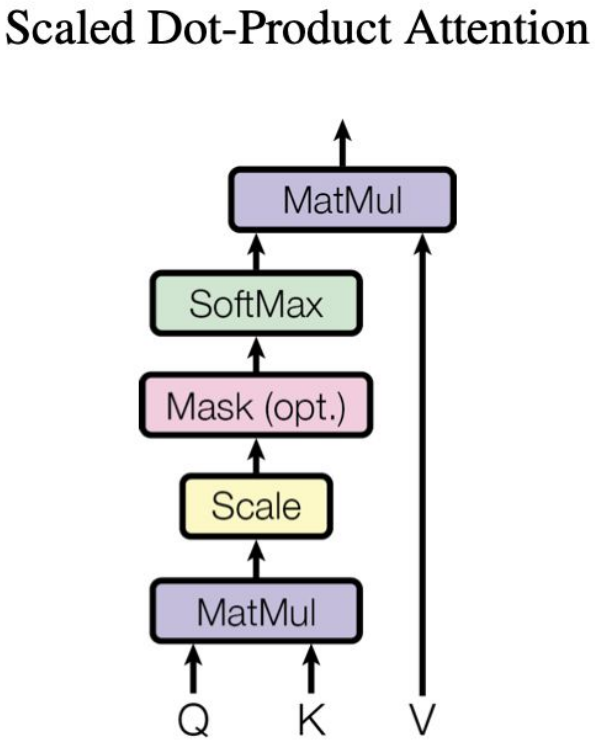
While the linear transformations are the same across different positions, they use different parameters from layer to layer. Another way of describing this is as two convolutions with kernel size 1. The dimensionality of input and output is $d_{\text{model}} = 512$, and the inner-layer has dimensionality $d_{ff} = 2048$.

scaled dot product attention

> first of all, what is a simple attention function?

> an attention function is a mapping between the queries and the key-value pairs.

> the output of the attention function is calculated as weighted sum of the values, where the weight assigned to the values is calculated by a compatibility function of the query with the corresponding key.

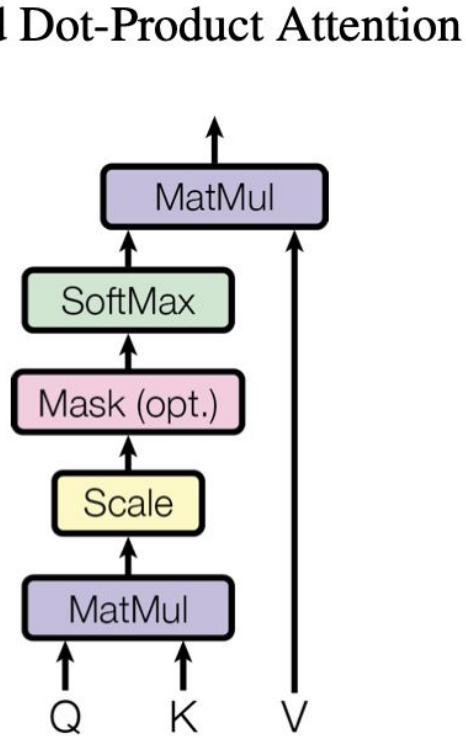


scaled dot product attention

> the inputs to the scaled dot product are the keys and the queries.

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

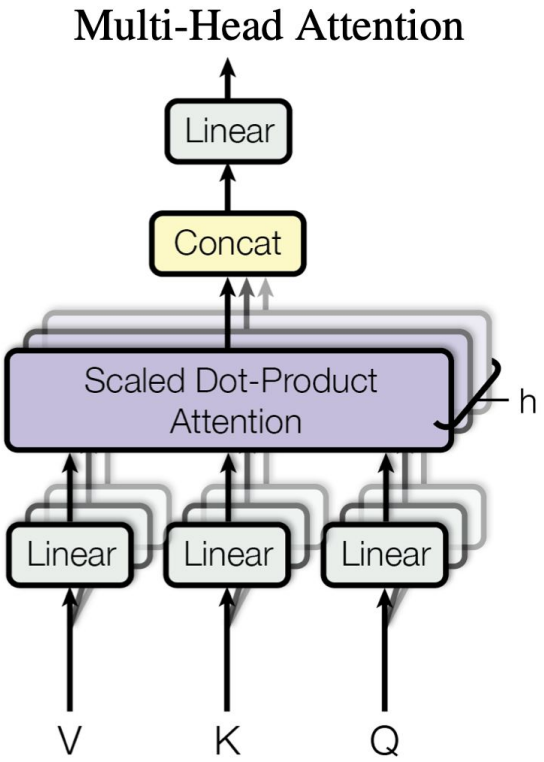
> the $\sqrt{d_k}$ is the scaling factor which helps the softmax from going into regions where it may suffer from having very low gradients.



multi-head attention

> now instead of performing a single scaled-dot product attention function we use multi-head attention.

> it was found that if the queries, keys and values are linearly projected ‘h’ times with different, learned linear projections, it allows the model to jointly attend to the information from different representation subspaces at different positions.



multi-head attention

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_\text{h})W^O$$
$$\text{where head}_\text{i} = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

Where the projections are parameter matrices $W_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$ and $W^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$.

In this work we employ $h = 8$ parallel attention layers, or heads. For each of these we use $d_k = d_v = d_{\text{model}}/h = 64$. Due to the reduced dimension of each head, the total computational cost is similar to that of single-head attention with full dimensionality.

positional encodings

> due to there being no recurrences nor convolutions, it was important to provide the model with the notion of relative / absolute positioning of the tokens, and so the positional encodings are added to the model with the input embeddings at the bottom of both the encoder and the decoder stacks.

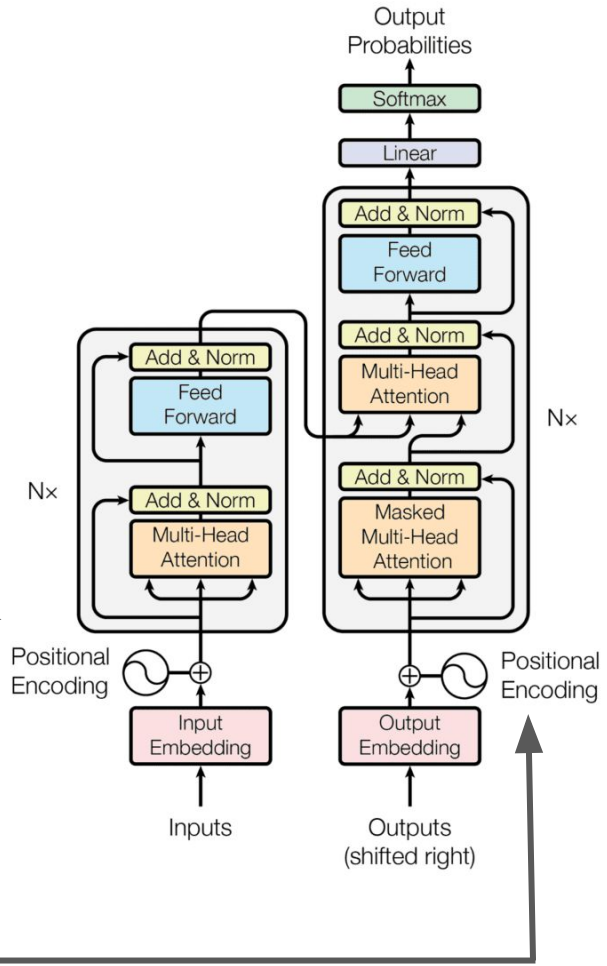
$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

where pos is the position and i is the dimension. That is, each dimension of the positional encoding corresponds to a sinusoid. The wavelengths form a geometric progression from 2π to $10000 \cdot 2\pi$. We chose this function because we hypothesized it would allow the model to easily learn to attend by relative positions, since for any fixed offset k, PE_{pos+k} can be represented as a linear function of PE_{pos}.

We also experimented with using learned positional embeddings [9] instead, and found that the two versions produced nearly identical results (see Table 3 row (E)). We chose the sinusoidal version because it may allow the model to extrapolate to sequence lengths longer than the ones encountered during training.

positional encodings

remember these?



some limitations in transformer models

> as proposed in the paper : Deja Vu : Contextual Sparsity for Efficient LLMs at Inference Time

---

Deja Vu: Contextual Sparsity for Efficient LLMs at Inference Time

---

Zichang Liu [1] **Jue Wang** [2] **Tri Dao** [3] **Tianyi Zhou** [4] **Binhang Yuan** [5] **Zhao Song** [6] **Anshumali Shrivastava** [1]
Ce Zhang [5] **Yuandong Tian** [7] **Christopher Ré** [3] **Beidi Chen** [8 7]

approach

> to reduce inference time in a LLM, the natural approach is one of sparsity.

> but existing methods at the time don’t really help a lot in this case, they usually require :

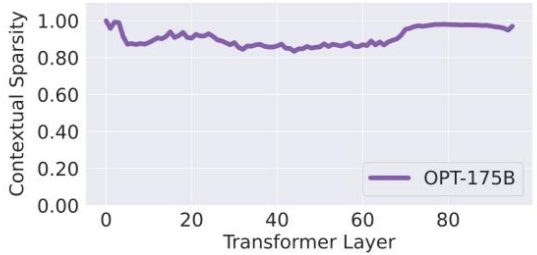
- > costly retraining,
- > have to forgo LLM’s in-context learning ability
- > or maybe even after doing a bunch of stuff, the wall clock time still doesn’t yield a significant speedup.

approach

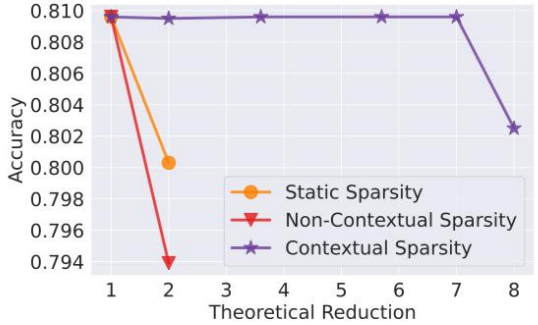
> the authors then hypothesized contextual sparsity :

> **there exist input-dependent sets of attention heads and MLP parameters which yield approximately the same output as the dense model for a given input.**

approach



(a) Contextual Sparsity



(b) Accuracy-Efficiency Trade-offs

Figure 1. (1) LLMs have up to 85% contextual sparsity for a given input. (2) Contextual sparsity has much better efficiency-accuracy trade-offs (up to 7×) than non-contextual sparsity or static sparsity.

approach

> previous problems with the pruning and sparsity methods as encountered were :

> **only applicable to small scale models**

> **it is challenging to find sparsity the preserves the in-context learning ability of the LLM**

> we can maybe discuss the ideas in this paper next time as we'll now move onto even exciting things!

> in the meanwhile you can check the "Attention is all you need " paper out for yourself at : https://arxiv.org/pdf/231(

> that is all for the transformers recap!

> some useful resources to dive more into the paper / references to this slide are :

> The Illustrated Transformer :

https://jalammar.github.io/illustrated-transformer/

> Peter Bloem's Transformer from Scratch :

https://peterbloem.nl/blog/transformers

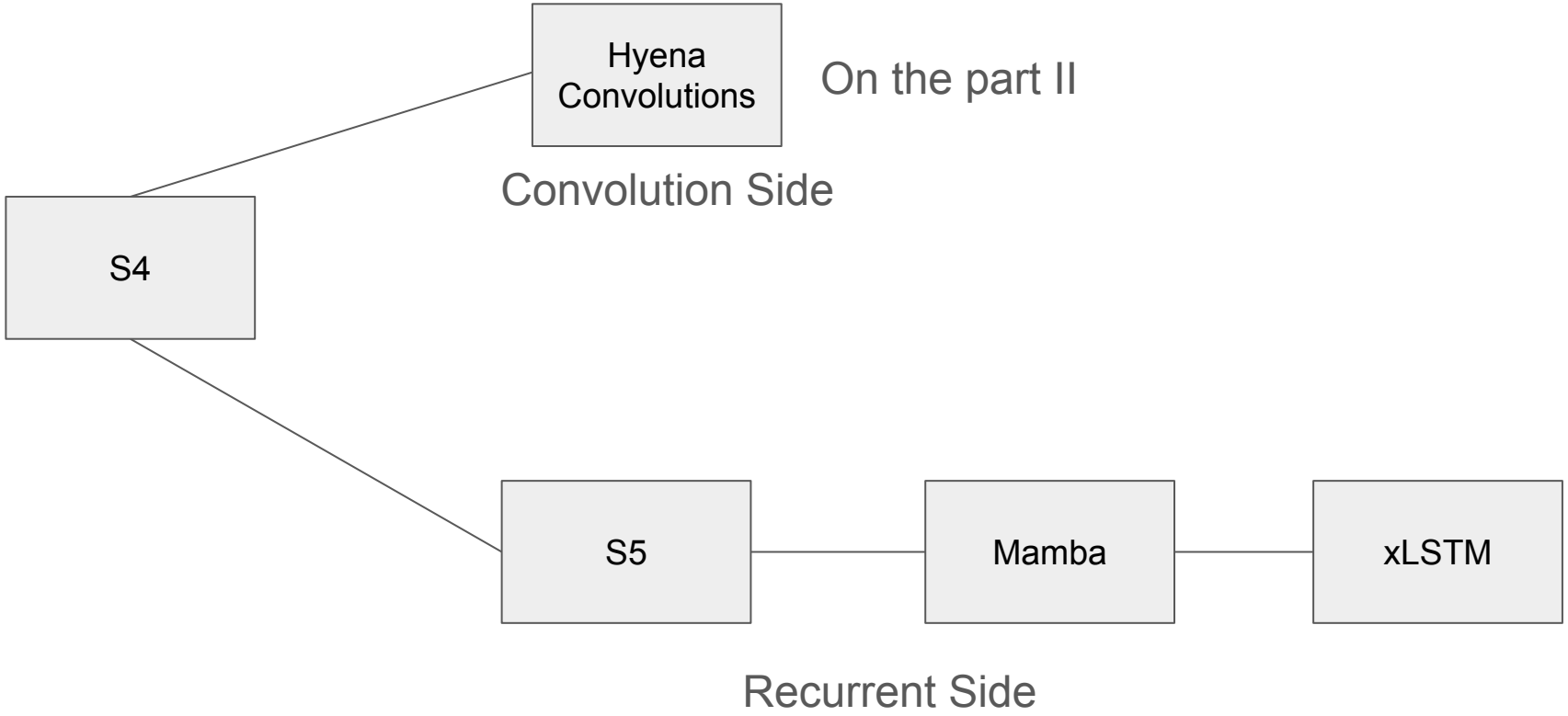
> FSDL 2021 :

https://fullstackdeeplearning.com/spring2021/lecture-4/

TRANSFORMERS RECAP

The End

Stepping Back from the Transformers

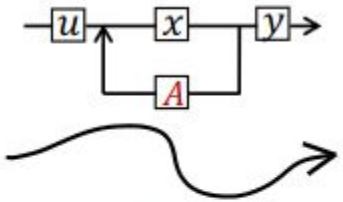


Efficiently Modeling Long Sequences with Structured State Spaces

Albert Gu, Karan Goel, and Christopher Ré

Department of Computer Science, Stanford University

{albertgu,krng}@stanford.edu, chrismre@cs.stanford.edu

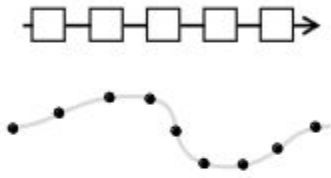


$$\dot{x} = Ax + Bu$$
$$y = Cx + Du$$

Continuous State Space

$$A = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 2 & 0 \\ 1 & 3 & 3 \end{bmatrix}$$

Long-Range Dependencies



$$x = \bar{A}x + \bar{B}u$$
$$y = \bar{C}x + \bar{D}u$$

$$y = \bar{K} * u$$

Fast Discrete Representations

Algorithm point of view on SSMs

- Take parameters A,B,C,D (either by predetermined way like A in s4 hippo matrix or Parameterize it (Mamba models))
- Descritize the parameter either Bilinear transformation, exponentiating the delta parameter (Zero Order Hold) but convert the continuous distribution into Discrete distribution because In recurrent connections we feed Discretized input even in the case of Classical RNNs
- Use those discretized parameters to apply a particular C to the output and estimate it as Y
- So in LSTM terms we say Y is equivalent to C from output gate and x_k is equivalent to hidden layer (h_k)

2.3 Discrete-time SSM: The Recurrent Representation

To be applied on a discrete input sequence $(u_0, u_1, \dots)$ instead of continuous function $u(t)$, (1) must be discretized by a **step size** Δ that represents the resolution of the input. Conceptually, the inputs u_k can be viewed as sampling an implicit underlying continuous signal $u(t)$, where $u_k = u(k\Delta)$.

To discretize the continuous-time SSM, we follow prior work in using the bilinear method [43], which converts the state matrix $\boldsymbol{A}$ into an approximation $\overline{\boldsymbol{A}}$. The discrete SSM is

$$\begin{aligned} x_k &= \overline{\boldsymbol{A}} x_{k-1} + \overline{\boldsymbol{B}} u_k & \overline{\boldsymbol{A}} &= (\boldsymbol{I} - \Delta/2 \cdot \boldsymbol{A})^{-1}(\boldsymbol{I} + \Delta/2 \cdot \boldsymbol{A}) \\ y_k &= \overline{\boldsymbol{C}} x_k & \overline{\boldsymbol{B}} &= (\boldsymbol{I} - \Delta/2 \cdot \boldsymbol{A})^{-1}\Delta \boldsymbol{B} \qquad \overline{\boldsymbol{C}} = \boldsymbol{C}. \end{aligned} \tag{3}$$

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

2.4 Training SSMs: The Convolutional Representation

The recurrent SSM (3) is not practical for training on modern hardware due to its sequentiality. Instead, there is a well-known connection between linear time-invariant (LTI) SSMs such as (1) and continuous convolutions. Correspondingly, (3) can actually be written as a discrete convolution.

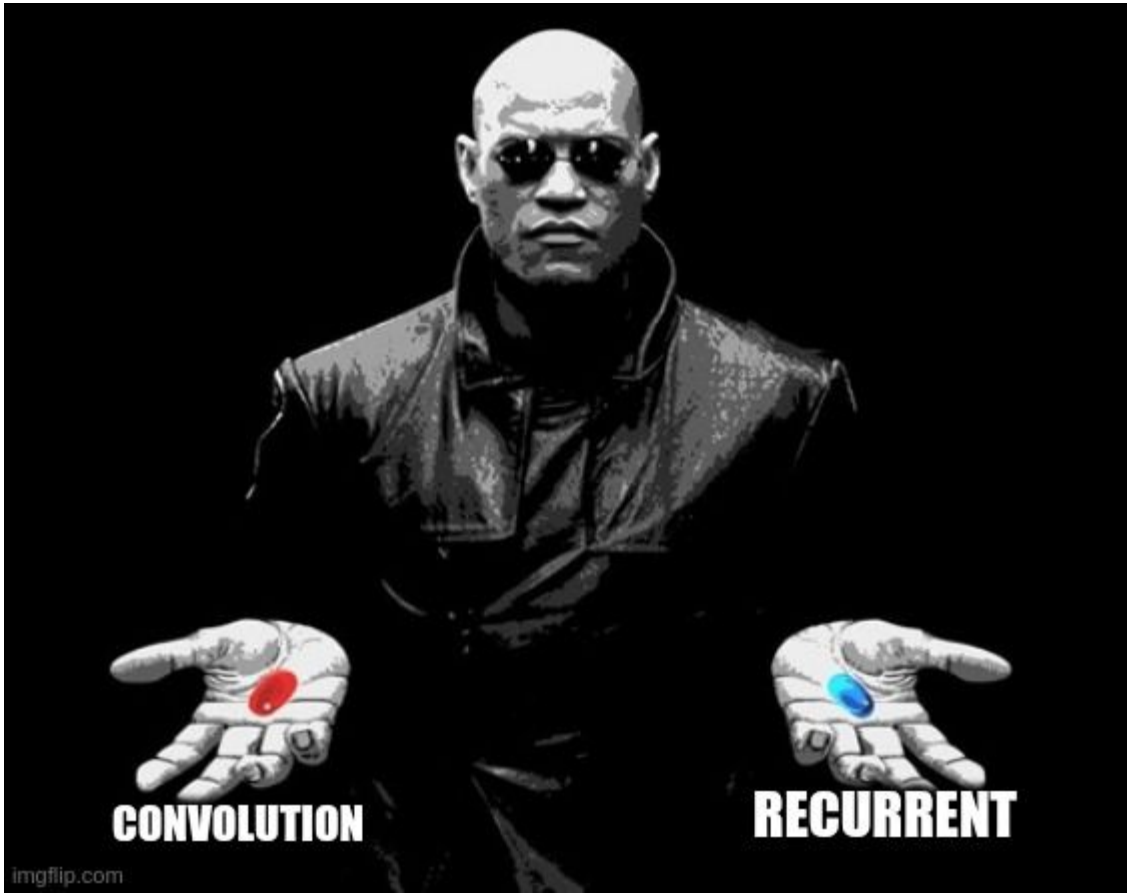
For simplicity let the initial state be $x_{-1} = 0$. Then unrolling (3) explicitly yields

$$
\begin{aligned}
x_0 &= \overline{\boldsymbol{B}} u_0 & x_1 &= \overline{\boldsymbol{A}\boldsymbol{B}} u_0 + \overline{\boldsymbol{B}} u_1 & x_2 &= \overline{\boldsymbol{A}}^2 \overline{\boldsymbol{B}} u_0 + \overline{\boldsymbol{A}\boldsymbol{B}} u_1 + \overline{\boldsymbol{B}} u_2 & \dots \\
y_0 &= \overline{\boldsymbol{C}\boldsymbol{B}} u_0 & y_1 &= \overline{\boldsymbol{C}\boldsymbol{A}\boldsymbol{B}} u_0 + \overline{\boldsymbol{C}\boldsymbol{B}} u_1 & y_2 &= \overline{\boldsymbol{C}\boldsymbol{A}}^2 \overline{\boldsymbol{B}} u_0 + \overline{\boldsymbol{C}\boldsymbol{A}\boldsymbol{B}} u_1 + \overline{\boldsymbol{C}\boldsymbol{B}} u_2 & \dots
\end{aligned}
$$

This can be vectorized into a convolution (4) with an explicit formula for the convolution kernel (5).

$$
\begin{aligned}
y_k &= \overline{\boldsymbol{C}\boldsymbol{A}}^k \overline{\boldsymbol{B}} u_0 + \overline{\boldsymbol{C}\boldsymbol{A}}^{k-1} \overline{\boldsymbol{B}} u_1 + \cdots + \overline{\boldsymbol{C}\boldsymbol{A}\boldsymbol{B}} u_{k-1} + \overline{\boldsymbol{C}\boldsymbol{B}} u_k \\
y &= \overline{\boldsymbol{K}} * u.
\end{aligned} \tag{4}
$$

$$
\overline{\boldsymbol{K}} \in \mathbb{R}^L := \mathcal{K}_L(\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}}, \overline{\boldsymbol{C}}) := \left(\overline{\boldsymbol{C}\boldsymbol{A}}^i \overline{\boldsymbol{B}} \right)_{i \in [L]} = (\overline{\boldsymbol{C}\boldsymbol{B}}, \overline{\boldsymbol{C}\boldsymbol{A}\boldsymbol{B}}, \dots, \overline{\boldsymbol{C}\boldsymbol{A}}^{L-1} \overline{\boldsymbol{B}}). \tag{5}
$$


CONVOLUTION
RECURRENT
imgflip.com

Going on the recurrent path Now (part 1)

- S5
- Mamba (S6)
- Bi directional Mamba (vision Mamba)

H3 ,Vss (to look)

S5 Major Insights

Now A matrix is parameterized with continuous Framework which contains a Combination of both Eigenvalues and Eigenvectors

$A = V\Lambda V^{-1}$, where $\Lambda \in CP \times P$ and $V \in CP \times P$

$\Lambda = e\Lambda\Delta$ is discredited with exponential function with step size of delta and delta is learnable

Which make it selective scanning from previous inputs

S5 paper does Selective Scanning

$$\frac{\mathrm{d}\mathbf{V}^{-1}\mathbf{x}(t)}{\mathrm{d}t} = \mathbf{\Lambda}\mathbf{V}^{-1}\mathbf{x}(t) + \mathbf{V}^{-1}\mathbf{B}\mathbf{u}(t). \tag{4}$$

Defining $\tilde{\mathbf{x}}(t) = \mathbf{V}^{-1}\mathbf{x}(t)$, $\tilde{\mathbf{B}} = \mathbf{V}^{-1}\mathbf{B}$, and $\tilde{\mathbf{C}} = \mathbf{C}\mathbf{V}$ gives a reparameterized system,

$$\frac{\mathrm{d}\tilde{\mathbf{x}}(t)}{\mathrm{d}t} = \mathbf{\Lambda}\tilde{\mathbf{x}}(t) + \tilde{\mathbf{B}}\mathbf{u}(t), \qquad \mathbf{y}(t) = \tilde{\mathbf{C}}\tilde{\mathbf{x}}(t) + \mathbf{D}\mathbf{u}(t). \tag{5}$$

This is a linear SSM with a diagonal state matrix. This diagonalized system can be discretized with a timescale parameter $\Delta \in \mathbb{R}_+$ using the ZOH method to give another diagonalized system with parameters

$$\overline{\mathbf{\Lambda}} = e^{\mathbf{\Lambda}\Delta}, \quad \overline{\mathbf{B}} = \mathbf{\Lambda}^{-1}(\overline{\mathbf{\Lambda}} - \mathbf{I})\tilde{\mathbf{B}}, \quad \overline{\mathbf{C}} = \tilde{\mathbf{C}}, \quad \overline{\mathbf{D}} = \mathbf{D}. \tag{6}$$

In practice, we use a vector of learnable timescale parameters $\mathbf{\Delta} \in \mathbb{R}^P$ (see Section 4.3) and restrict the feedthrough matrix $\mathbf{D}$ to be diagonal. The S5 layer therefore has the learnable parameters: $\tilde{\mathbf{B}} \in \mathbb{C}^{P\times H}$, $\tilde{\mathbf{C}} \in \mathbb{C}^{H\times P}$, $\mathrm{diag}(\mathbf{D}) \in \mathbb{R}^H$, $\mathrm{diag}(\mathbf{\Lambda}) \in \mathbb{C}^P$, and $\mathbf{\Delta} \in \mathbb{R}^P$.

Now Mamba block!!

Mamba addresses is selectively scannable issue and introduced gated convolution with Silu activation function

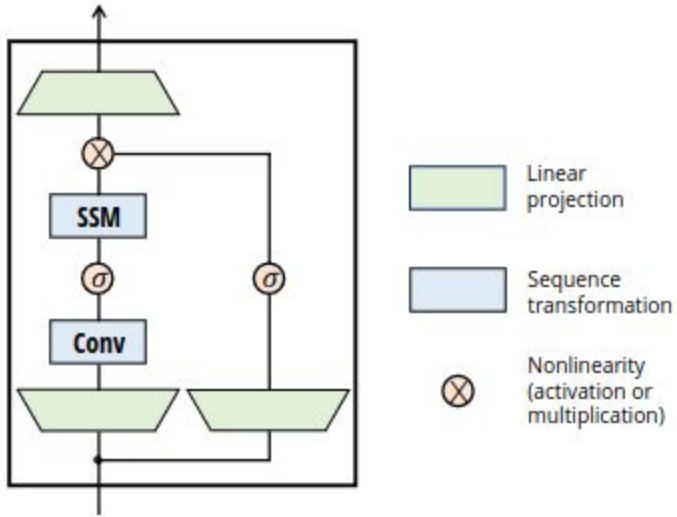
With that we could simulate transformer rather than having Group of SSM blocks. And Attention could be Linearly Written by transformations which is discussed in next slide

Thus we could have attention in a recurrent form Manner Thus making Mamba Block equivalent to transformers architecturally

Mamba

Mamba is inspired from linear attention where we could compute Attention recurrently by doing small modification in the Original attention equation.

It uses 2 Linear layers and convolution layer followed by S6 SSM(which is introduced in this paper) it gets transformer like architecture.


SSM
Conv
Linear projection
Sequence transformation
Nonlinearity (activation or multiplication)

S6 Algorithm SSM part of mamba

Algorithm 2 SSM + Selection (S6)

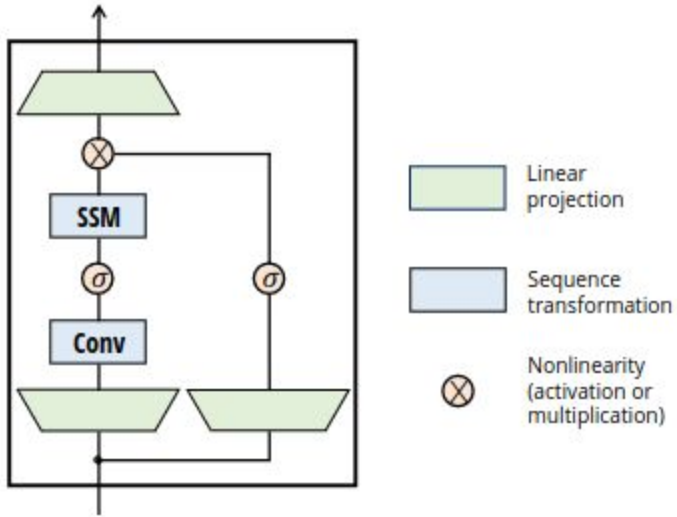
Input: $x : (\mathtt{B}, \mathtt{L}, \mathtt{D})$
Output: $y : (\mathtt{B}, \mathtt{L}, \mathtt{D})$

1: $\boldsymbol{A} : (\mathtt{D}, \mathtt{N}) \leftarrow$ Parameter
▷ Represents structured $N \times N$ matrix
2: $\boldsymbol{B} : (\mathtt{B}, \mathtt{L}, \mathtt{N}) \leftarrow s_B(x)$
3: $\boldsymbol{C} : (\mathtt{B}, \mathtt{L}, \mathtt{N}) \leftarrow s_C(x)$
4: $\Delta : (\mathtt{B}, \mathtt{L}, \mathtt{D}) \leftarrow \tau_\Delta(\text{Parameter} + s_\Delta(x))$
5: $\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}} : (\mathtt{B}, \mathtt{L}, \mathtt{D}, \mathtt{N}) \leftarrow \text{discretize}(\Delta, \boldsymbol{A}, \boldsymbol{B})$
6: $y \leftarrow \text{SSM}(\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}}, \boldsymbol{C})(x)$
▷ Time-varying: recurrence (*scan*) only
7: **return** y

Discretization of Parameters in SSMs

Discretization. The first stage transforms the "continuous parameters" $(\Delta, \boldsymbol{A}, \boldsymbol{B})$ to "discrete parameters" $(\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}})$ through fixed formulas $\overline{\boldsymbol{A}} = f_A(\Delta, \boldsymbol{A})$ and $\overline{\boldsymbol{B}} = f_B(\Delta, \boldsymbol{A}, \boldsymbol{B})$, where the pair (f_A, f_B) is called a discretization rule. Various rules can be used such as the zero-order hold (ZOH) defined in equation (4).

$$\overline{\boldsymbol{A}} = \exp(\Delta \boldsymbol{A}) \qquad \overline{\boldsymbol{B}} = (\Delta \boldsymbol{A})^{-1}(\exp(\Delta \boldsymbol{A}) - \boldsymbol{I}) \cdot \Delta \boldsymbol{B} \tag{4}$$


SSM
Conv
Linear projection
Sequence transformation
Nonlinearity (activation or multiplication)

Mamba on Vision

1. Vision Mamba uses both forward and backward Mamba blocks on Image data.
2. Vision Mamba merely works as ViT does the same as it
3. Forward and backward way of using SSM modules.
4. Forward way just computes Normal SSM in flattened 2d patches
5. Those Flattened patches are reversed and computed in different SSM called backward SSM

Vision Mamba Algorithm

Algorithm 1 Vim Block Process

Require: token sequence $\mathbf{T}_{l-1}$: $(\mathtt{B}, \mathtt{M}, \mathtt{D})$
Ensure: token sequence $\mathbf{T}_{l}$: $(\mathtt{B}, \mathtt{M}, \mathtt{D})$

/* normalize the input sequence $\mathbf{T}'_{l-1}$ */
$\mathbf{T}'_{l-1}$: $(\mathtt{B}, \mathtt{M}, \mathtt{D}) \leftarrow \mathbf{Norm}(\mathbf{T}_{l-1})$
$\mathbf{x}$: $(\mathtt{B}, \mathtt{M}, \mathtt{E}) \leftarrow \mathbf{Linear}^{\mathbf{x}}(\mathbf{T}'_{l-1})$
$\mathbf{z}$: $(\mathtt{B}, \mathtt{M}, \mathtt{E}) \leftarrow \mathbf{Linear}^{\mathbf{z}}(\mathbf{T}'_{l-1})$
/* process with different direction */
for o in {forward, backward} **do**
  $\mathbf{x}'_o$: $(\mathtt{B}, \mathtt{M}, \mathtt{E}) \leftarrow \mathbf{SiLU}(\mathbf{Conv1d}_o(\mathbf{x}))$
  $\mathbf{B}_o$: $(\mathtt{B}, \mathtt{M}, \mathtt{N}) \leftarrow \mathbf{Linear}^{\mathbf{B}}_o(\mathbf{x}'_o)$
  $\mathbf{C}_o$: $(\mathtt{B}, \mathtt{M}, \mathtt{N}) \leftarrow \mathbf{Linear}^{\mathbf{C}}_o(\mathbf{x}'_o)$
  /* softplus ensures positive $\mathbf{\Delta}_o$ */
  $\mathbf{\Delta}_o$: $(\mathtt{B}, \mathtt{M}, \mathtt{E}) \leftarrow \log(1 + \exp(\mathbf{Linear}^{\mathbf{\Delta}}_o(\mathbf{x}'_o) + \mathbf{Parameter}^{\mathbf{\Delta}}_o))$
  /* shape of $\mathbf{Parameter}^{\mathbf{A}}_o$ is $(\mathtt{E}, \mathtt{N})$ */
  $\overline{\mathbf{A}}_o$: $(\mathtt{B}, \mathtt{M}, \mathtt{E}, \mathtt{N}) \leftarrow \mathbf{\Delta}_o \bigotimes \mathbf{Parameter}^{\mathbf{A}}_o$
  $\overline{\mathbf{B}}_o$: $(\mathtt{B}, \mathtt{M}, \mathtt{E}, \mathtt{N}) \leftarrow \mathbf{\Delta}_o \bigotimes \mathbf{B}_o$
  $\mathbf{y}_o$: $(\mathtt{B}, \mathtt{M}, \mathtt{E}) \leftarrow \mathbf{SSM}(\overline{\mathbf{A}}_o, \overline{\mathbf{B}}_o, \mathbf{C}_o)(\mathbf{x}'_o)$
end for
/* get gated $\mathbf{y}_o$ */
$\mathbf{y}'_{forward}$: $(\mathtt{B}, \mathtt{M}, \mathtt{E}) \leftarrow \mathbf{y}_{forward} \bigodot \mathbf{SiLU}(\mathbf{z})$
$\mathbf{y}'_{backward}$: $(\mathtt{B}, \mathtt{M}, \mathtt{E}) \leftarrow \mathbf{y}_{backward} \bigodot \mathbf{SiLU}(\mathbf{z})$
/* residual connection */
$\mathbf{T}_l$: $(\mathtt{B}, \mathtt{M}, \mathtt{D}) \leftarrow \mathbf{Linear}^{\mathbf{T}}(\mathbf{y}'_{forward} + \mathbf{y}'_{backward}) + \mathbf{T}_{l-1}$
Return: $\mathbf{T}_l$

Vision Mamba Description