Mamba

Transformers can be different:

- During *qkv*, calculating the attention score of each token (relative to every other) is O(n^2) according to the length of the context window.
- The quadratically-growing key-value cache needs to be stored alongside the model during inference.
- having a context window at all is rather limiting

If we can do this better, we can hopefully be better at speech and video (context-heavy tasks)

What are State Space Models?

We can think of SSMs as blackbox mapping u(t) → y(t):

- A, B, C, and D are learnable latent parameters
- And x(t) is a solution to the linear ODE that represents the latent representation

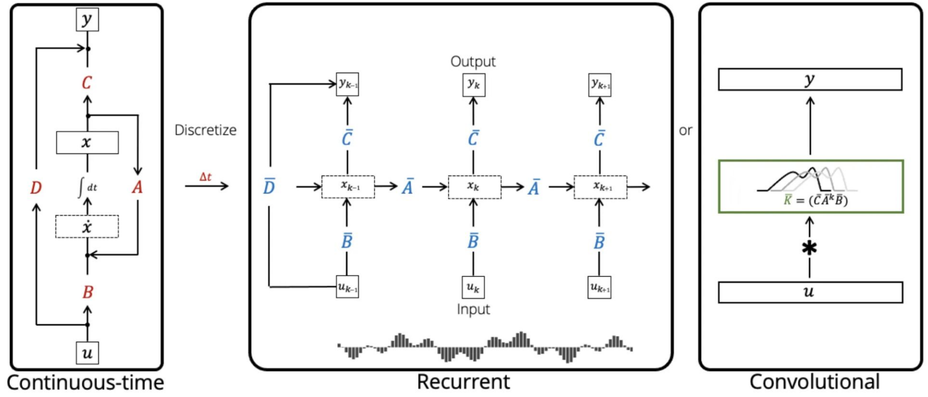
$$
\begin{aligned}
\mathrm{x}'(\mathrm{t}) &= \mathrm{Ax}(\mathrm{t}) + \mathrm{Bu}(\mathrm{t}) \\
\mathrm{y}(\mathrm{t}) &= \mathrm{Cx}(\mathrm{t}) + \mathrm{Du}(\mathrm{t})
\end{aligned}
$$

SSMs: Continuous, Recurrent, and Convolutional

SSMs transform into different views:

Recurrent:

$$
\begin{aligned}
x_k &= A x_{k-1} + B u_k \\
y_k &= C x_k + D u_k
\end{aligned}
$$



Convolutional:

$$
x_0 = \bar{B}u_0 \quad x_1 = \bar{A}\bar{B}u_0 + \bar{B}u_1 \quad x_2 = \bar{A}^2\bar{B}u_0 + \bar{A}\bar{B}u_1 + \bar{B}u_2 \quad \cdots
$$

Let’s drop the state space model idea

Let’s say we’re just looking at the recurrent version: RNNs

Pros:

- No context window (unlike convolutional view)
- Efficient constant time inference (unlike the continuous view),

Cons:

- Not parallelizable
- Exploding/vanishing gradients (if we truly want a large effective context)

Linear RNNs

How to Parallelize?

Normal RNNs are too complicated: let’s remove the activation function.

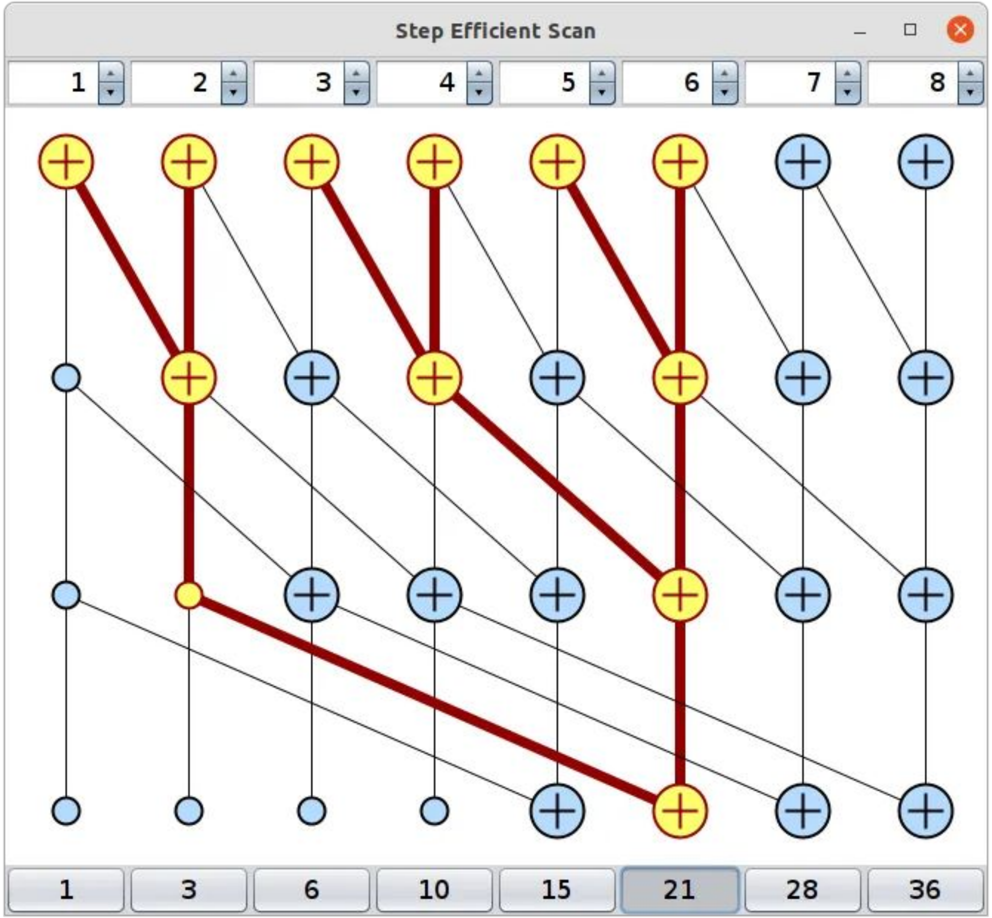
$$h_t = f_W(h_{t-1}, x_t)$$

$$f_W(h, x) = W_h h + W_x x$$

Blelloch

Blelloch scan allows us to find the prefix sum of an array very quickly

This is only because addition is associative.



Associative RNN Iteration

This function turns out to be associative, allowing us to iterate over all inputs in the RNN quickly (W_h and W_x folded into W)

$$\mathbf{f}((\mathbf{W}_1, \mathbf{x}_1), (\mathbf{W}_2, \mathbf{x}_2)) = (\mathbf{W}_1\mathbf{W}_2, \mathbf{W}_1\mathbf{x}_1 + \mathbf{x}_2)$$

Associative RNN Iteration

Note this also means we have to cache W_i W_j … W_k which is d x d for each position in the array.

That's a lot — but luckily we can diagonalize W and simply store diagonal elements

$$f((W_1, x_1), (W_2, x_2)) = (W_1 W_2, W_1 x_1 + x_2)$$

We're now parallelizable!

We are now parallelizable in O(n log(n)) time!

Cool facts

- P and P^{-1} are learned by a model to not have to deal with and matrix inverting, while adding more expressivity
- We still want nonlinearity so we can add a nonlinear layer after doing all the recurrent iterations (which will be much quicker) — just like the dense layer after attention

But exploding gradients?

Initialize initial weights very close to 1

$$w = e^{-e^{a}} e^{ib}, e^{-e^{a}} \sim \mathrm{Uniform}([0.999, 1.0]), b \sim \mathrm{Uniform}([0, \frac{\pi}{10}])$$

And multiply all inputs by a very small number because our model is sensitive!

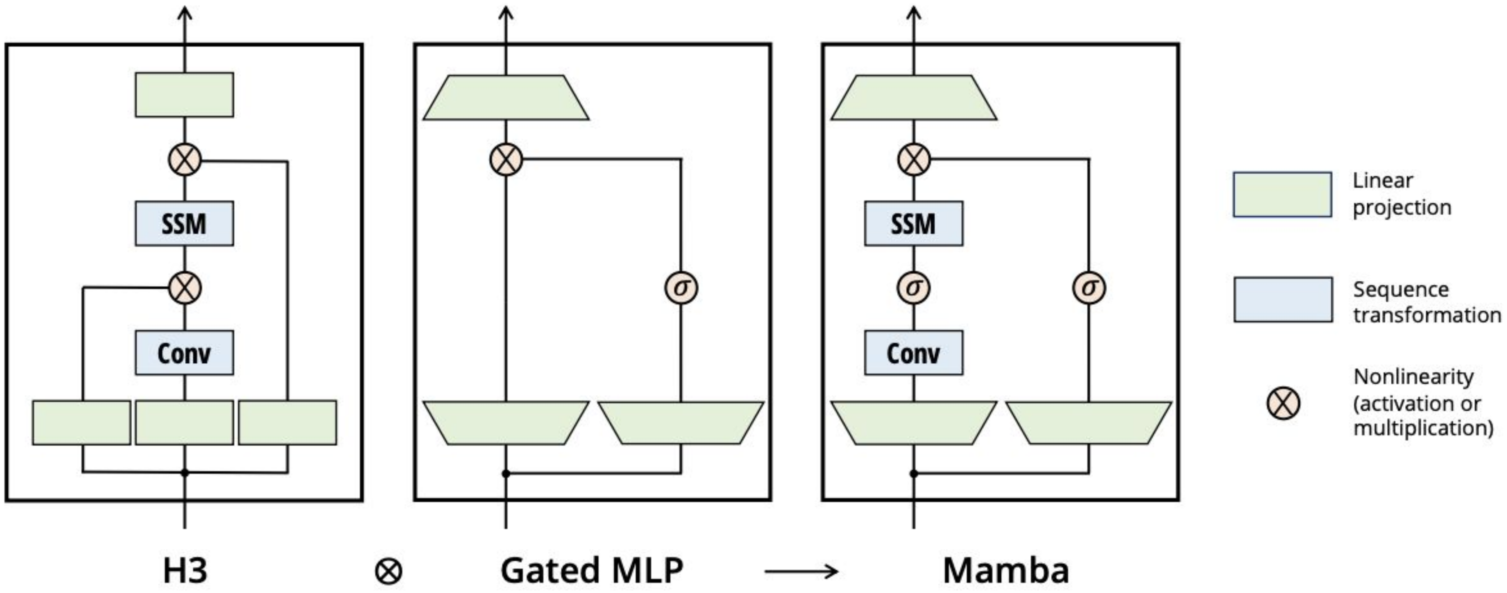
$$x := \Delta x$$

$$\Delta = \sqrt{1 - e^{-e^{a}}}.$$

Mamba

Selective SSM: Adding a Gate

RNNs have to hold too much info in h_t. We want to be selective on what to hold



Selectivity Implemented

Remember that this is essentially an RNN

Here — we define functions that parameterize W_h and W_x based on inputs themselves.

Algorithm 1 SSM (S4)

Input: $x : (\mathtt{B}, \mathtt{L}, \mathtt{D})$
Output: $y : (\mathtt{B}, \mathtt{L}, \mathtt{D})$
$\boldsymbol{A} : (\mathtt{D}, \mathtt{N}) \leftarrow \mathsf{Parameter}$
▷ Represents structured $N \times N$ matrix
$\boldsymbol{B} : (\mathtt{D}, \mathtt{N}) \leftarrow \mathsf{Parameter}$
$\boldsymbol{C} : (\mathtt{D}, \mathtt{N}) \leftarrow \mathsf{Parameter}$
$\Delta : (\mathtt{D}) \leftarrow \tau_\Delta(\mathsf{Parameter})$
$\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}} : (\mathtt{D}, \mathtt{N}) \leftarrow \mathsf{discretize}(\Delta, \boldsymbol{A}, \boldsymbol{B})$
$y \leftarrow \mathsf{SSM}(\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}}, \boldsymbol{C})(x)$
▷ Time-invariant: recurrence or convolution
return y

Algorithm 2 SSM + Selection (S6)

Input: $x : (\mathtt{B}, \mathtt{L}, \mathtt{D})$
Output: $y : (\mathtt{B}, \mathtt{L}, \mathtt{D})$
$\boldsymbol{A} : (\mathtt{D}, \mathtt{N}) \leftarrow \mathsf{Parameter}$
▷ Represents structured $N \times N$ matrix
$\boldsymbol{B} : (\mathtt{B}, \mathtt{L}, \mathtt{N}) \leftarrow s_B(x)$
$\boldsymbol{C} : (\mathtt{B}, \mathtt{L}, \mathtt{N}) \leftarrow s_C(x)$
$\Delta : (\mathtt{B}, \mathtt{L}, \mathtt{D}) \leftarrow \tau_\Delta(\mathsf{Parameter} + s_\Delta(x))$
$\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}} : (\mathtt{B}, \mathtt{L}, \mathtt{D}, \mathtt{N}) \leftarrow \mathsf{discretize}(\Delta, \boldsymbol{A}, \boldsymbol{B})$
$y \leftarrow \mathsf{SSM}(\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}}, \boldsymbol{C})(x)$
▷ Time-varying: recurrence (*scan*) only
return y

(Introduction of L = length → time-dependent. Input-dependency → batches different)

Selectivity

"Selecting functions" are chosen where

$$s_B(x) = \text{Linear}_N(x),\ s_C(x) = \text{Linear}_N(x),\ s_\Delta(x) = \text{Broadcast}_D(\text{Linear}_1(x)),\ \text{and}\ \tau_\Delta = \text{softplus},$$

such that given with A = -1 and B = 1, the gate at each head ends up looking like

$$g_t = \sigma(\text{Linear}(x_t))$$
$$h_t = (1 - g_t)h_{t-1} + g_t x_t$$

Eloquent, isn't it?