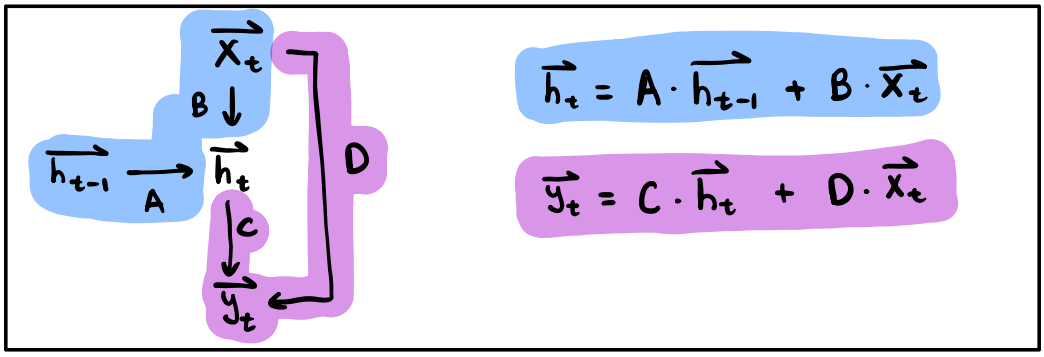
STATE SPACE MODELS

Why should I care?

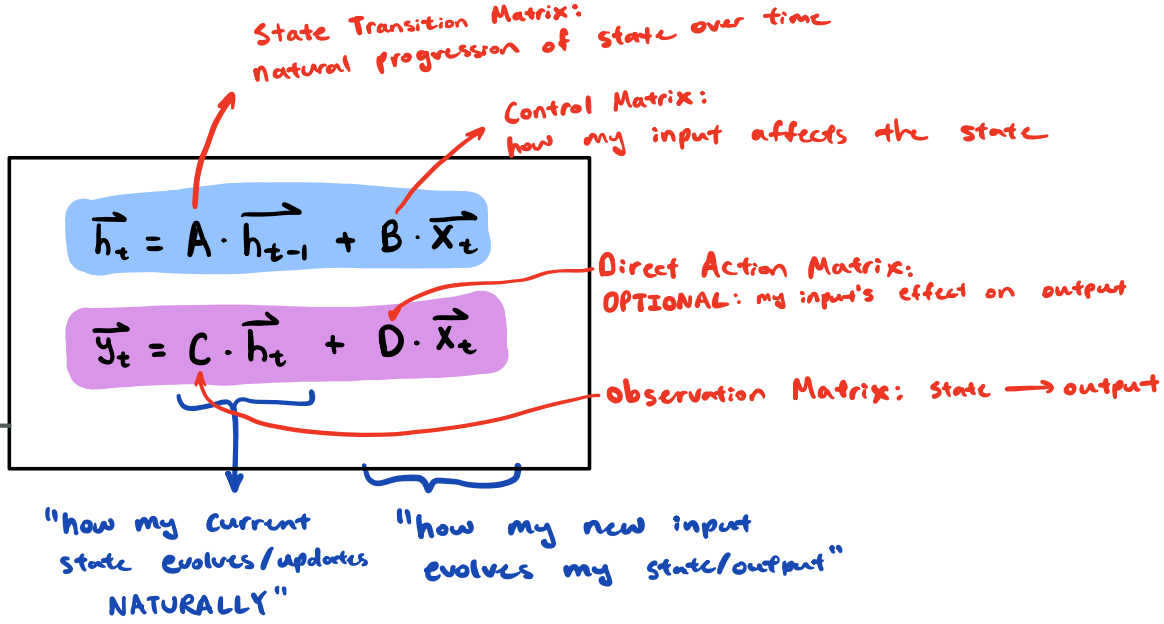
Pitch: constant processing time for new inputs — no dependence on the sequence length!

What even are SSMs, anyway?

Boils down to a single image:



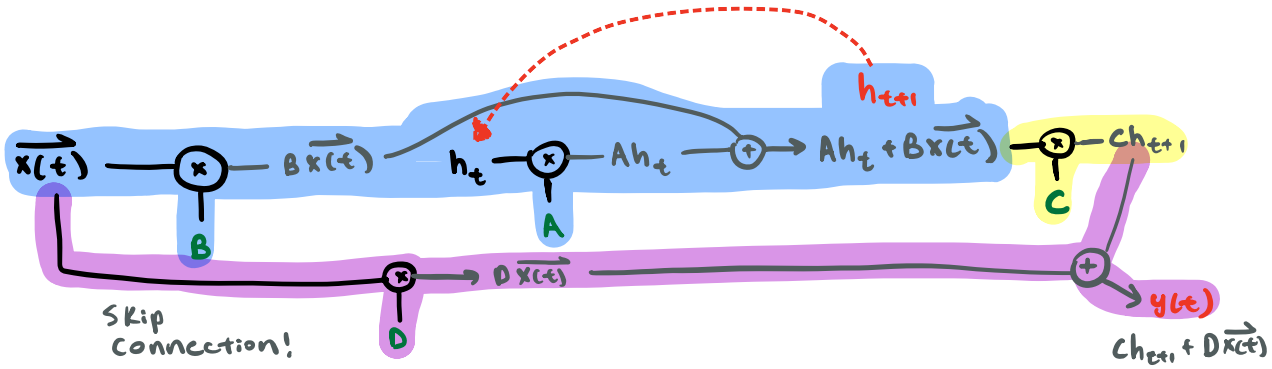
Adding labels to things:



$\vec{x(t)}$ = an input signal e.g. $\vec{x(0)} = \begin{bmatrix} 0 \\ 0 \\ 10 \end{bmatrix}$, $\vec{x(1)} = \begin{bmatrix} 2 \\ 1 \\ 12 \end{bmatrix}$, $\vec{x(2)} = \begin{bmatrix} 5 \\ 3 \\ 11 \end{bmatrix}$

$\vec{h(t)}$ = my "context"/internal memory e.g. $[x, y, v_x, v_y, a_x, a_y]^T$

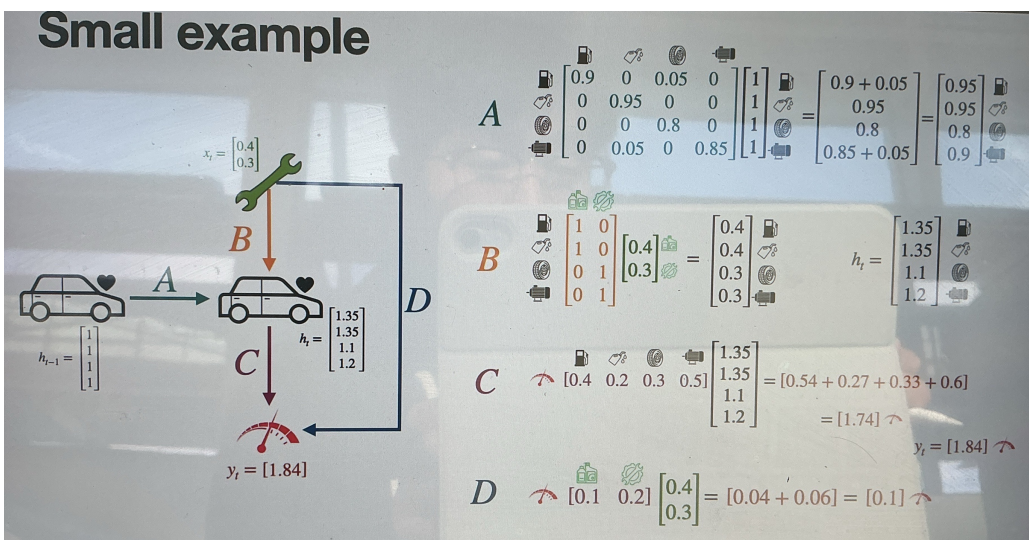
Here's the exact computation graph:



This is a dead-simple recursive relationship:

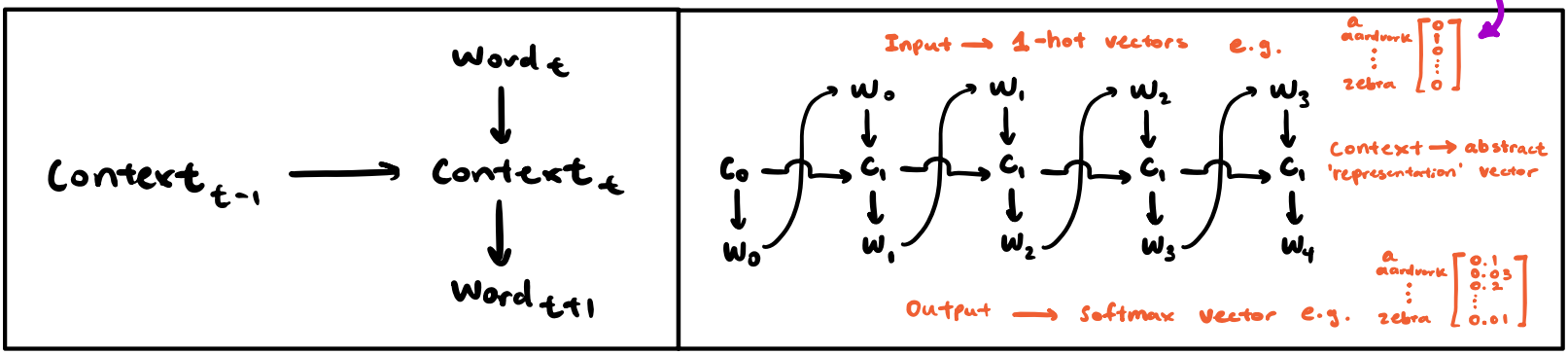
			output
$t=0$	given $\vec{x}_0, \vec{h}_0$:	$\vec{y}_0 = C\cdot\vec{h}_0 + D\cdot\vec{x}_0$ $\vec{h}_1 = A\cdot\vec{h}_0 + B\cdot\vec{x}_0$	$\vec{y}_0$
$t=1$	given $\vec{x}_1$:	$\vec{y}_1 = C\cdot\vec{h}_1 + D\cdot\vec{x}_1$ $\vec{h}_2 = A\cdot\vec{h}_1 + B\cdot\vec{x}_1$	$\vec{y}_1$
$t=2$	given $\vec{x}_2$:	$\vec{y}_2 = C\cdot\vec{h}_2 + D\cdot\vec{x}_2$ $\vec{h}_3 = A\cdot\vec{h}_2 + B\cdot\vec{x}_2$	$\vec{y}_2$

"State" is just a general idea: I like this video's analogy to cars:



SSMs can be used for language!

Very RNN-like...



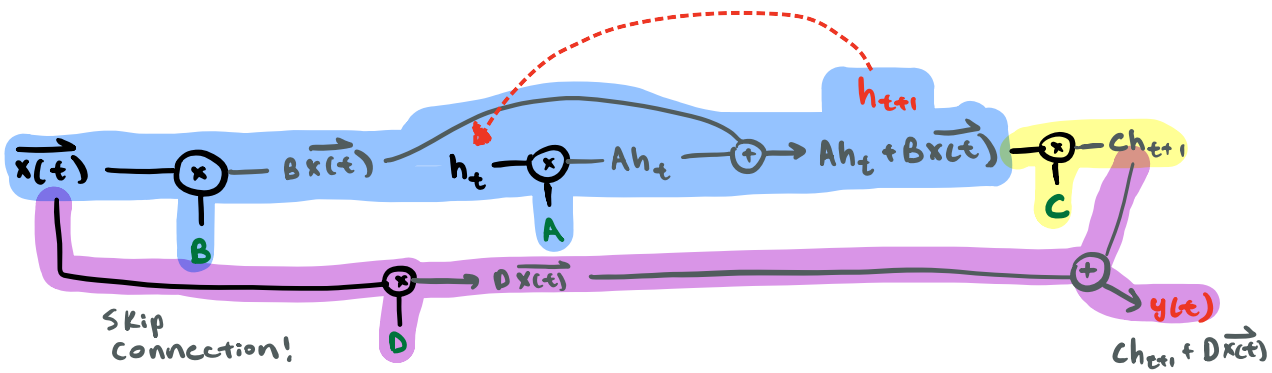
1. Embed your words!

The → [1.0, 0.8, ...]
Cat → [0.5, 3.0, ...]
Sat → [0.7, 9.0, ...]

2. Treat embeddings like input signal

The → [1.0, 0.8, ...] → $\vec{x}(0)$
Cat → [0.5, 3.0, ...] → $\vec{x}(1)$
Sat → [0.7, 9.0, ...] → $\vec{x}(2)$
⋮

3. Apply SSM computation



Wait, isn't this just an RNN?

YES! But it's a special subset of RNNs that are purely linear, & this gives it a VERY cool property.

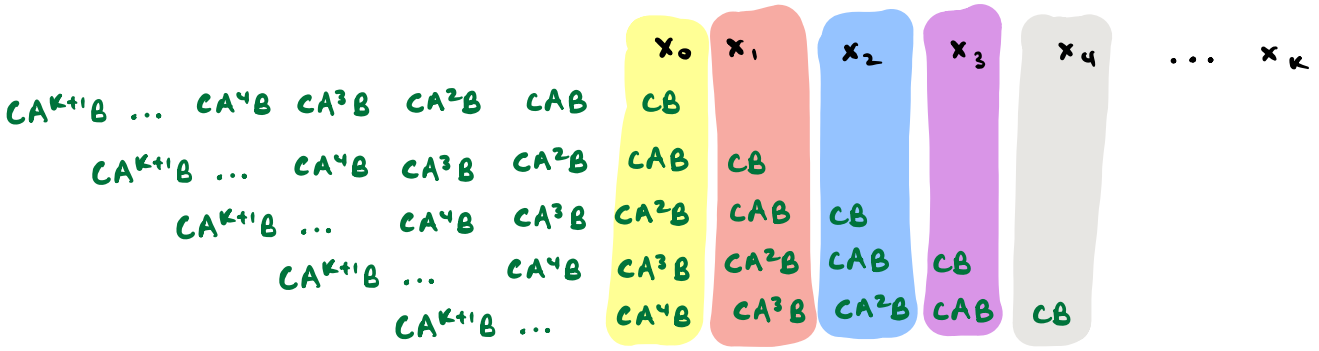
I gave this recurrence relation:

$$\vec{h}_t = A \cdot \vec{h}_{t-1} + B \cdot \vec{x}_t$$
$$\vec{y}_t = C \cdot \vec{h}_t + D \cdot \vec{x}_t$$

Let's just expand it out!

$$h_1 = Bx_1$$
$$h_2 = Ah_1 + Bx_2 = ABx_1 + Bx_2$$
$$h_3 = Ah_2 + Bx_3 = A^2Bx_1 + ABx_2 + Bx_3$$
$$\vdots$$
$$h_k = A^kBx_1 + A^{k-1}Bx_2 + \dots + ABx_k + Bx_{k+1}$$
$$y_k = C \cdot h_k = CA^kBx_1 + \dots + CBx_{k+1}$$

Looking at the coefficients, we can formulate this as a 1-D Convolution over the input sequence!



$$\underbrace{[CA^{k+1}B \dots CB]}_{\text{Kernel}} \overset{\text{Conv}}{*} \underbrace{[\vec{x}_1 \dots \vec{x}_{k+1}]}_{\text{input}} = [\vec{y}_1 \dots \vec{y}_{k+1}]$$

THIS IS HUGE: Convolutions are heavily optimized (they're really dense GEMMs under the hood). The linearity allows us to avoid the sequential bottleneck!

Worked Example:

$$\vec{x}_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \quad \vec{x}_2 = \begin{bmatrix} 0 \\ 3 \end{bmatrix} \qquad A = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix} \quad B = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

① Calculate the Kernel: ← Just GEMMs!

$$B = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \qquad AB = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix}$$

② Convolve! (also just GEMMs)

$$\left(\begin{bmatrix} 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix} \right) * \left(\begin{bmatrix} 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 3 \end{bmatrix} \right) = \left(\begin{bmatrix} 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0.5 \\ 2 \end{bmatrix} \right)$$

↑ These are the core ideas behind (older) SSMs like S4.

↳ You might have noticed matrices A & B stay constant ... which is odd! Our "filter" for context isn't super flexible.

↓

Up next ... **Mamba**: a **selective** SSM that can choose what context is important!