Transformers

→ powerful memory, we can access specific parts of the input via KV

→ parallelize during training

→ slow at test-time

→ quadratic complexity in L

RNNs:

→ memory bottleneck

→ not parallelizable at training time

→ fast at test-time

RNN:

$$h_t = f(\underbrace{W_h h_{t-1}}_{\text{hidden state transition}} + \underbrace{W_e c_t}_{\text{current input token}})$$

linear RNN:

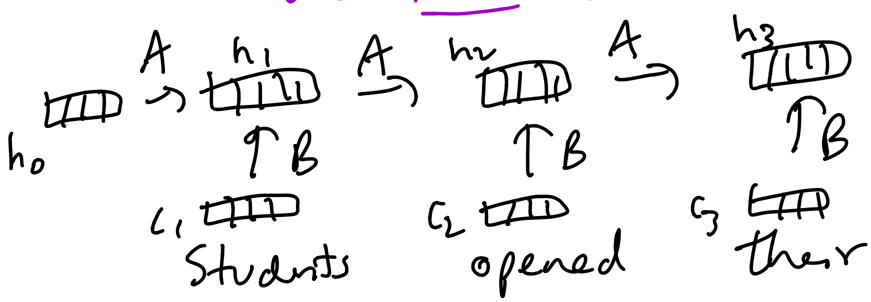
$$h_t = \overset{A}{\cancel{W_h}}\, \underbrace{h_{t-1}} + \overset{B}{\cancel{W_e}}\, c_t$$

⇒ no nonlinearities → lower expressivity

⇒ what happens if we put a nonlinear feedfwd layer on top?

$$z_t = \boxed{f}\left(\overbrace{Ch_t + Dc_t}\right)$$

↳ idea: all nonlinearities are above the linear recurrent layer



linear RNN:

$$h_1 = Bc_1$$

$$h_2 = Ah_1 + Bc_2$$

$$h_3 = Ah_2 + Bc_3$$

$$h_2 = ABc_1 + Bc_2$$

$$h_3 = A^2Bc_1 + ABc_2 + Bc_3$$

$$h_4 = A^3Bc_1 + A^2Bc_2 + \ldots$$

$$h_t = \sum_{k=0}^{t-1} A^k B c_{t-k}$$

how do we parallelize this?

$\Rightarrow$ treat this as a convolution, and design kernel

$$[B, AB, A^2B, \ldots, A^{k-1}B]$$

slide over sequence

$\Rightarrow$ in modern papers: <u>parallel scan</u>

let's consider the simple case of the cumulative sum operation

$$y_k = \sum_{i=1}^{k} x_i$$

3, 1, 7, 0, 4, 1, 6, 3

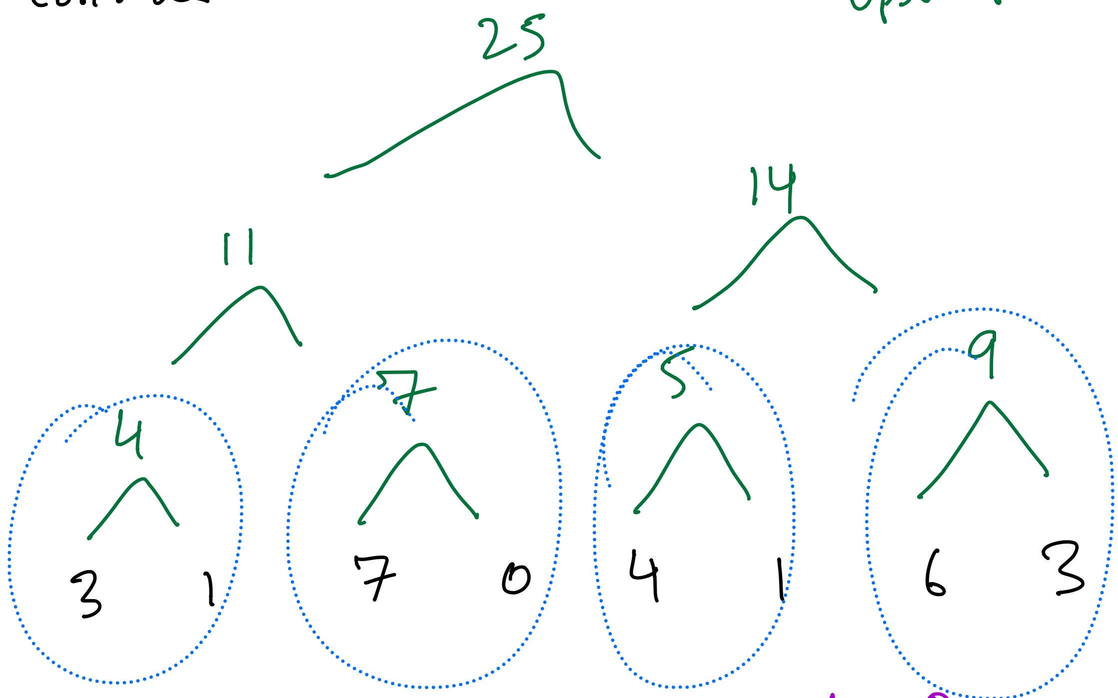
3, 4, 11, 11, 15, 16, 22, 25

sequentially:

$$y_t = x_t + y_{t-1}$$

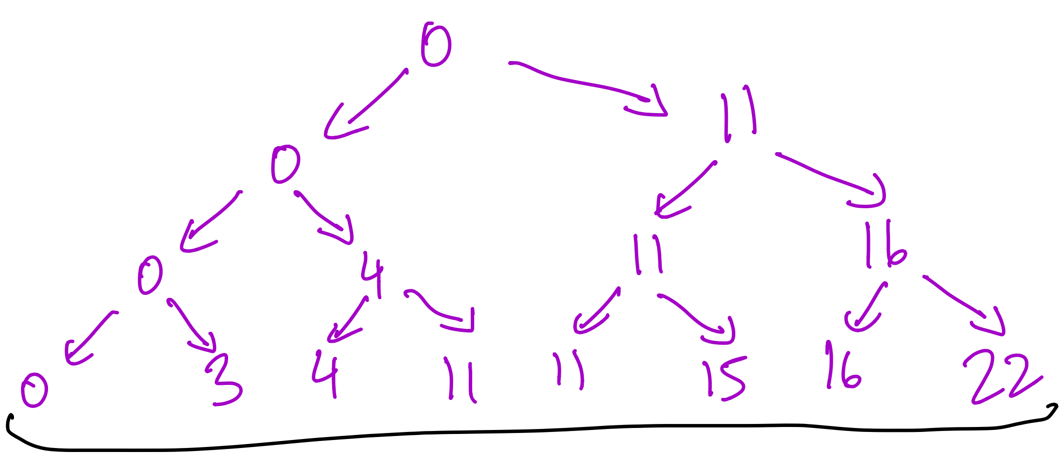
Consider:

upsweep



downsweep: - root starts at 0

- left child gets the value of the root
- right child gets the sum of the root + upsweep(left child)



3, 4, 11, 11, 15, 16, 22, 25

↳ we can parallelize these tree operations on diff processors

↳ can we make this work for linear RNN

instead of cum. sum, we want to compute:

$$h_t = A h_{t-1} + B c_t$$

consider the operation:

$$(a, b) \odot (a', b') = (a' a,\ a' b + b')$$

store matrix powers

h_t

$$(A^4, A^3Bc_0 + A^2Bc_1 + ABc_2 + Bc_3)$$

$$(A^2, ABc_0 + Bc_1) \qquad (A^2, ABc_2 + Bc_3)$$

$$(A, Bc_0) \qquad (A, Bc_1) \qquad (A, Bc_2) \qquad (A, Bc_3)$$