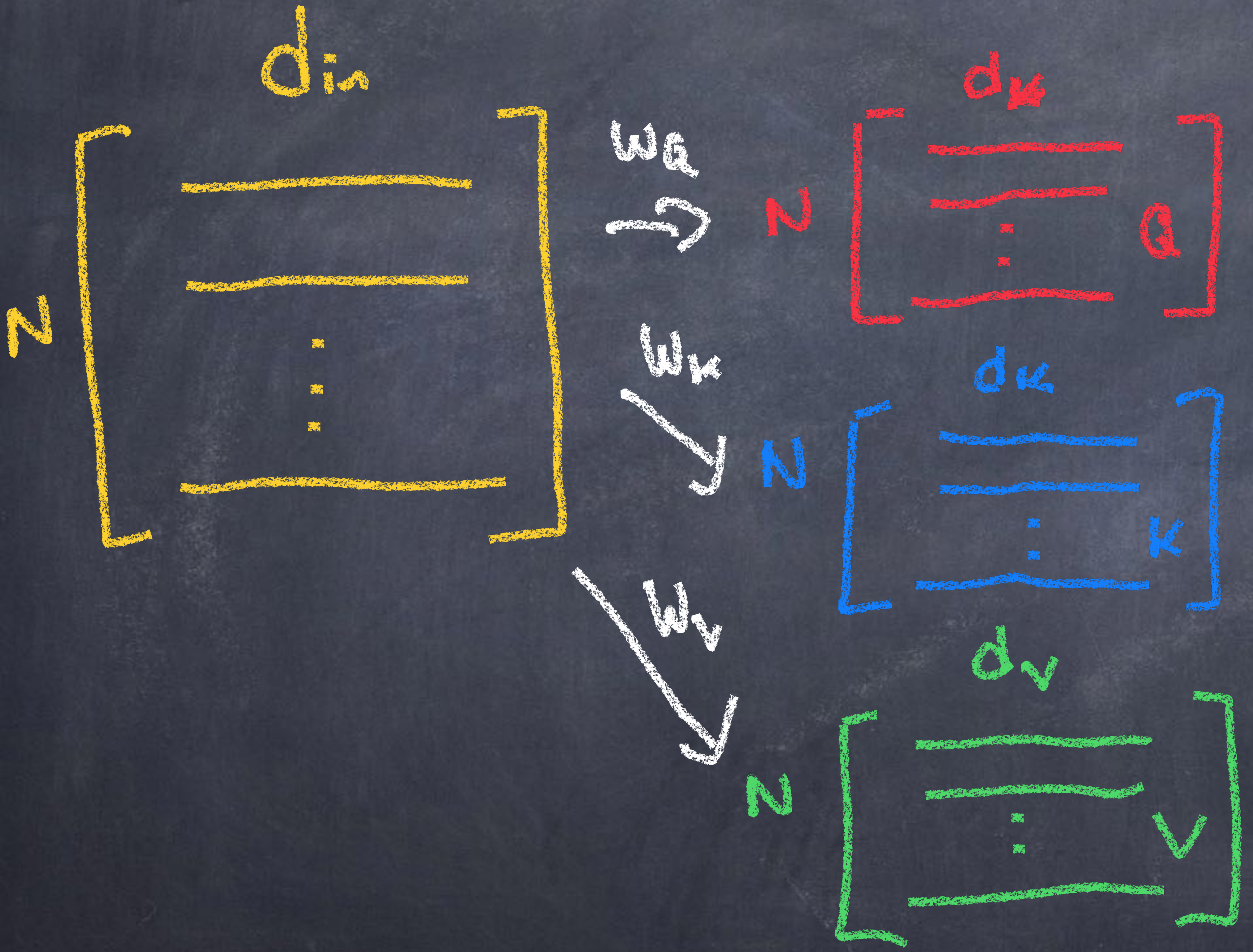
# Were RNNs all we needed?

# Context: Processing long sequences

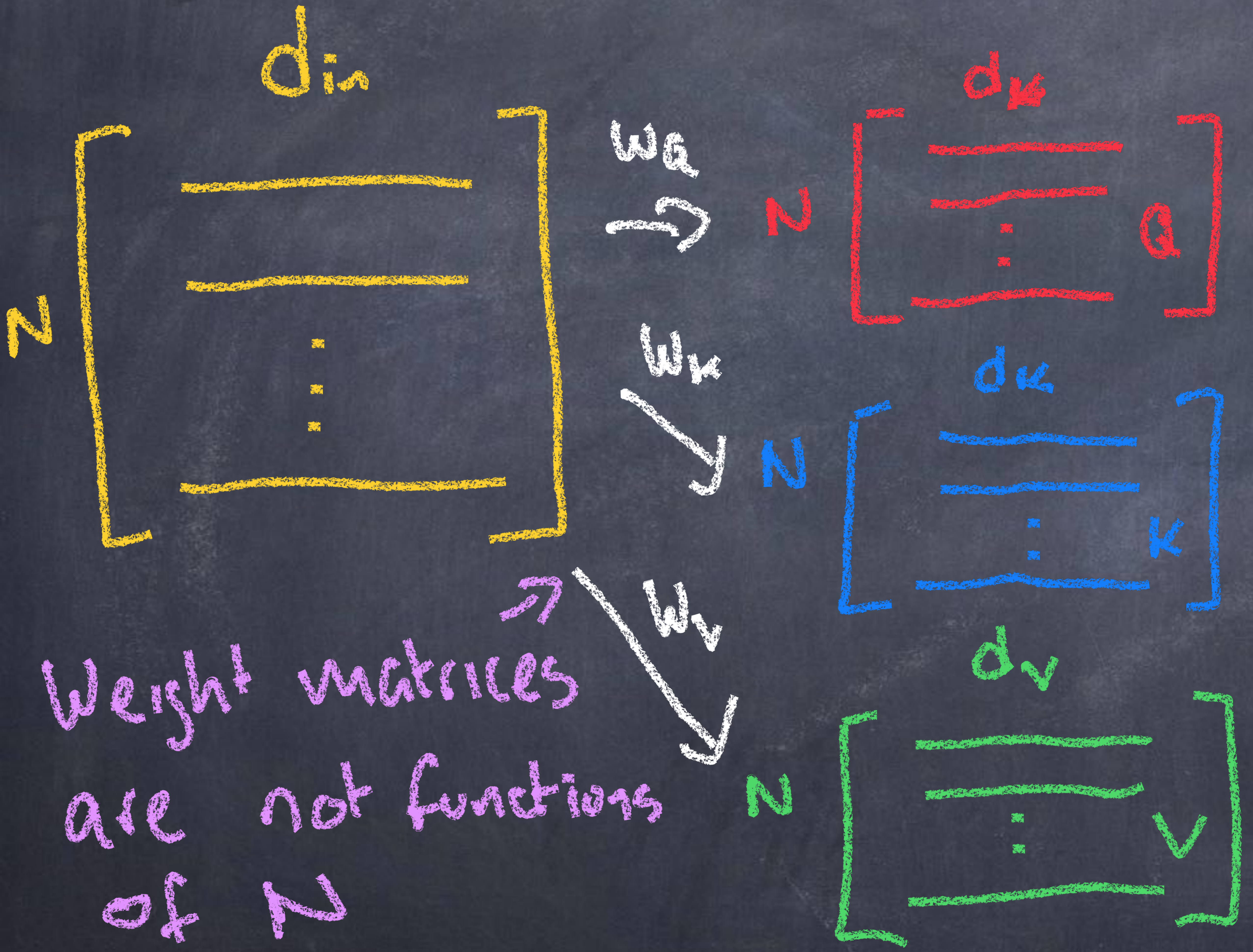
- Many sequence modeling tasks require long context windows
  - e.g. in a task like document summarization it helps to know everything about the document, not just a paragraph or two...
- Increasing the size of the context window is seen as a major research challenge
  - The limitation is a computational one

# Existing methods: transformer




$$\mathrm{Softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

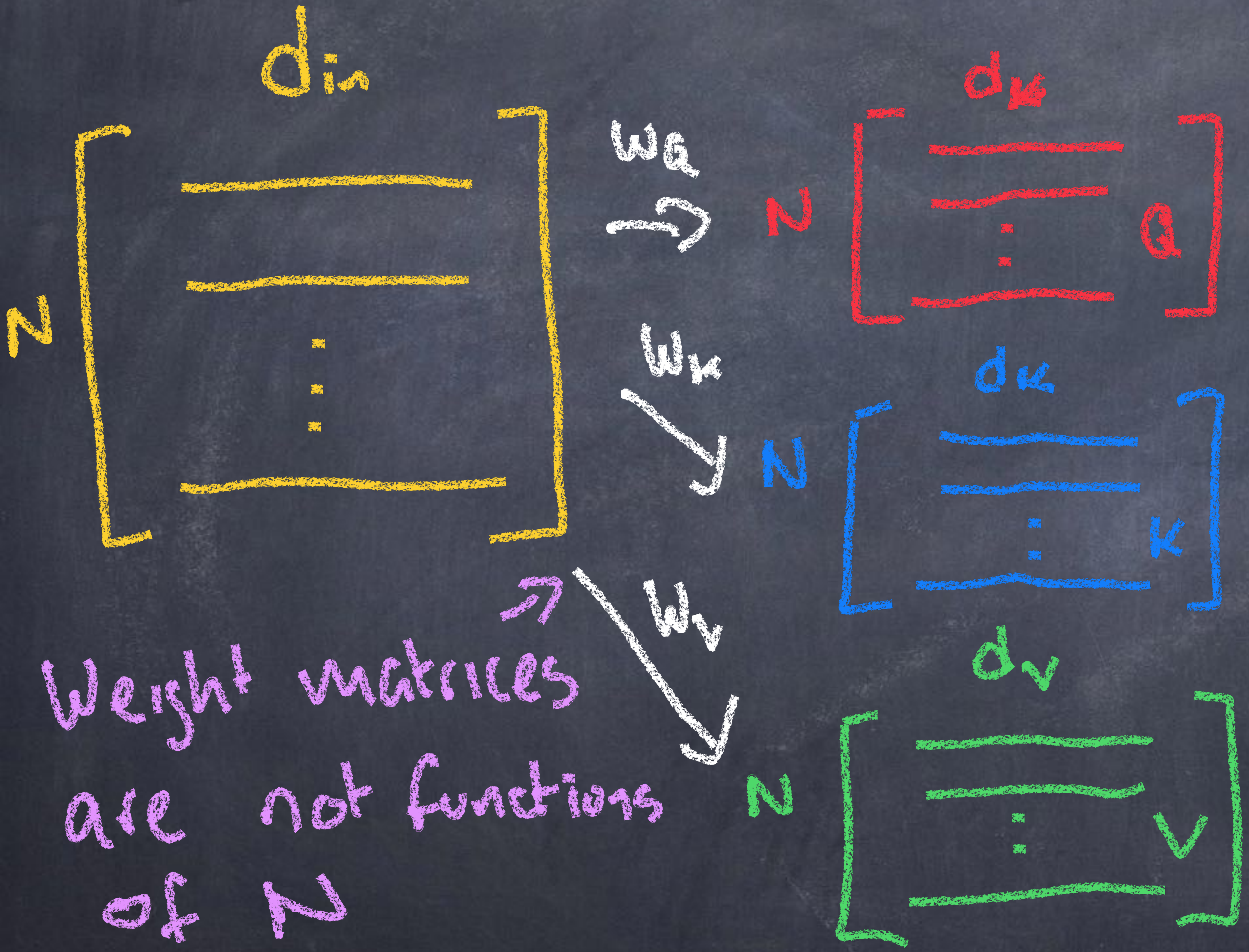
# Existing methods: transformer




$$\text{Softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Output is $N \times d_v$

# Existing methods: transformer




$$\text{Softmax}\left(\frac{Q\,K^T}{\sqrt{d_k}}\right)V$$

Output is $N \times d_v$

Transformer complexity is quadratic in N

# Existing methods: RNNs

- Older RNNs (Elman, LSTM, GRU) can in principle handle long sequences given enough compute resource
  - But, they are sequential (and thus slow), and might suffer other compute problems (vanishing gradient, etc) with really large contexts
    - They scale linearly with N
- Some hints that newer state-space models (s4, s6, Mamba, etc) are able to scale to long sequence tasks efficiently

# Can efficient, parallelisable RNNs be created?

# Background: parallel scan

- Idea: use multiple processors to efficiently compute

$$\left\{ \bigoplus_{i=1}^{k} u_i \right\}_{k=1}^{N}$$

- From a sequence

$$\left\{ u_k \right\}_{k=1}^{N}$$

- Where $\oplus$ is an associative operator

# Background: parallel scan
## e.g. cumulative sum

$1 + 2 = ?$

$1 + 2 + 3 = ?$

$1 + 2 + 3 + 4 = ?$

# Background: parallel scan
## e.g. cumulative sum

$1 + 2 = ?$

$1 + 2 + 3 = ?$

$1 + 2 + 3 + 4 = ?$

# Background: parallel scan

What about sequences $x_t = a_t x_{t-1} + b_t$ ?

# Background: parallel scan

What about sequences $x_t = a_t x_{t-1} + b_t$ ?

Take logs: $\log x_t = a_t^* + \log(x_0 + b_t^*)$

$$a_t^* = \sum_t^{\text{cum}} \log a_t$$

$$b_t^* = \sum_t^{\text{cum}} e^{\log b_t - a_t^*}$$

$$\Rightarrow x_t = e^{a_t^* + \log(x_0 + b_t^*)}$$

# Background: parallel scan

What about sequences $x_t = a_t x_{t-1} + b_t$?

Take logs: $\log x_t = a_t^* + \log(x_0 + b_t^*)$

$$a_t^* = \sum_t^{\text{cum}} \log a_t$$

$$b_t^* = \sum_t^{\text{cum}} e^{\log b_t - a_t^*}$$

} Both can be computed with parallel scan!

$$\Rightarrow x_t = e^{a_t^* + \log(x_0 + b_t^*)}$$

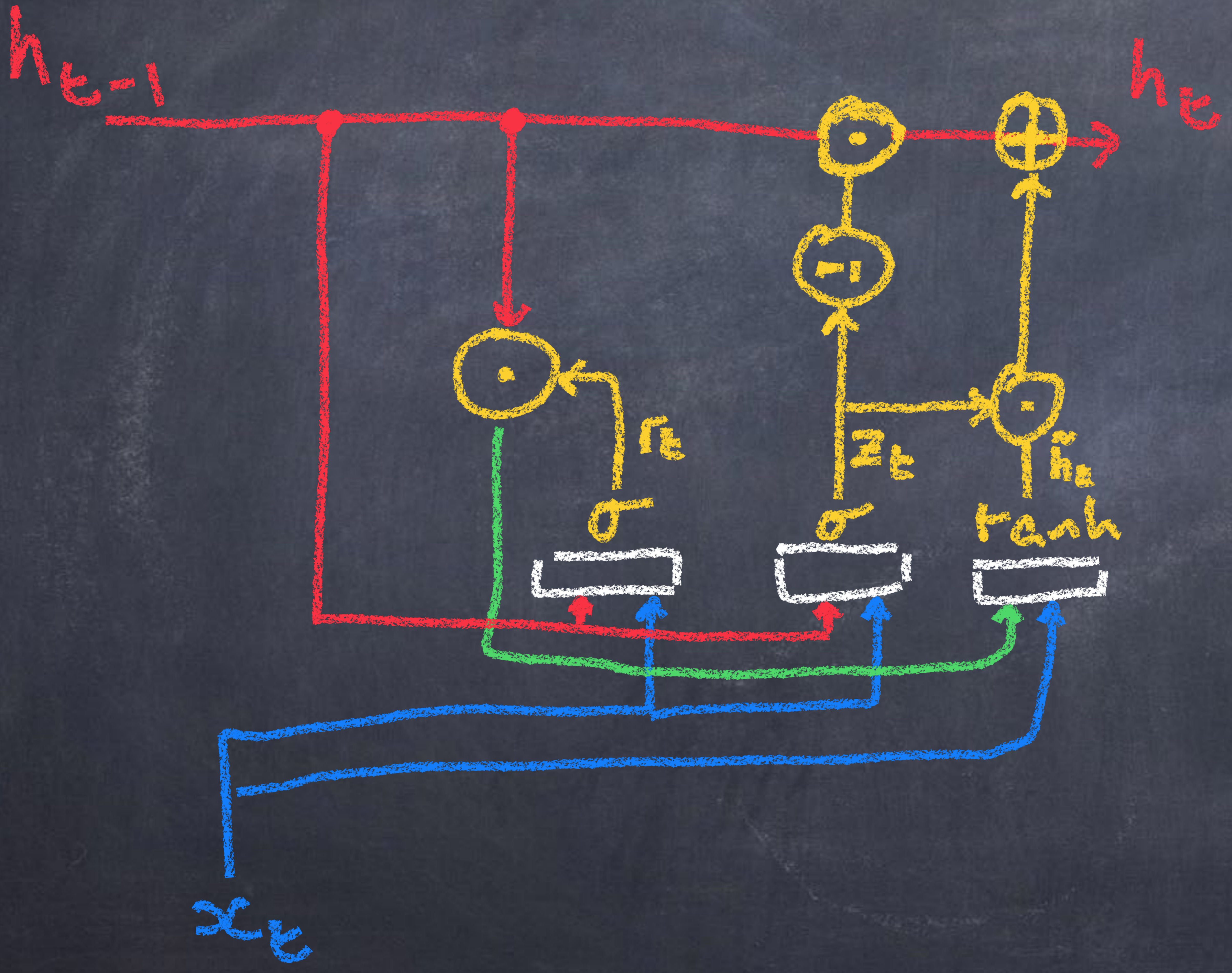
# Background: parallel scan

What about sequences $x_t = a_t x_{t-1} + b_t$ ?

$$\Longrightarrow x_t = e^{a_t^* + \log(x_0 + b_t^*)}$$

Note for implementation use log-cum-sum-exp trick for numerical stability

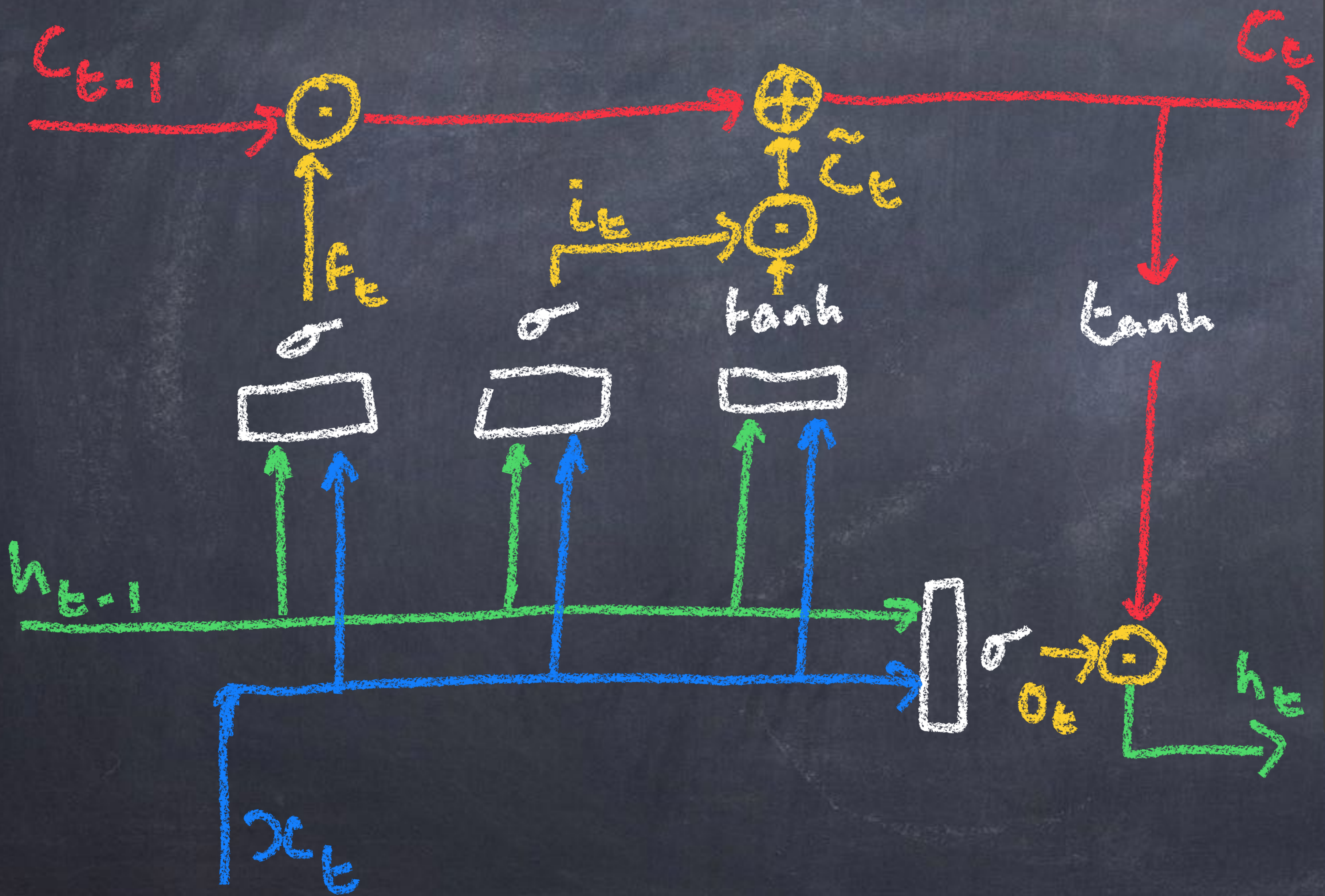
# Background: GRU




**GRU**

$$\boldsymbol{h}_t = (\boldsymbol{1} - \boldsymbol{z}_t) \odot \boldsymbol{h}_{t-1} + \boldsymbol{z}_t \odot \tilde{\boldsymbol{h}}_t$$
$$\boldsymbol{z}_t = \sigma(\mathrm{Linear}_{d_h}([\boldsymbol{x}_t, \boldsymbol{h}_{t-1}]))$$
$$\boldsymbol{r}_t = \sigma(\mathrm{Linear}_{d_h}([\boldsymbol{x}_t, \boldsymbol{h}_{t-1}]))$$
$$\tilde{\boldsymbol{h}}_t = \tanh(\mathrm{Linear}_{d_h}([\boldsymbol{x}_t, \boldsymbol{r}_t \odot \boldsymbol{h}_{t-1}]))$$

# Background: LSTM




**LSTM**

$$
\begin{aligned}
\boldsymbol{h}_t &= \boldsymbol{o}_t \odot \tanh(\boldsymbol{c}_t) \\
\boldsymbol{o}_t &= \sigma(\text{Linear}_{d_h}([\boldsymbol{x}_t, \boldsymbol{h}_{t-1}])) \\
\boldsymbol{c}_t &= \boldsymbol{f}_t \odot \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \odot \tilde{\boldsymbol{c}}_t \\
\boldsymbol{f}_t &= \sigma(\text{Linear}_{d_h}([\boldsymbol{x}_t, \boldsymbol{h}_{t-1}])) \\
\boldsymbol{i}_t &= \sigma(\text{Linear}_{d_h}([\boldsymbol{x}_t, \boldsymbol{h}_{t-1}])) \\
\tilde{\boldsymbol{c}}_t &= \tanh(\text{Linear}_{d_h}([\boldsymbol{x}_t, \boldsymbol{h}_{t-1}]))
\end{aligned}
$$

# Methodology

- Strip back existing RNNs so we can apply parallel scan
    - Remove hidden state dependencies on input/forget/update gates
- Remove constraints on output range
    - (Because by removing hidden state dependencies the vanishing/exploding gradient issues that resulted should also go)
    - Also ensure output is time-independent in scale
        - (i.e. stop output exploding/diminishing over time steps)

# Making minGRU

Parallel scan : $v_t = a_t \odot v_{t-1} + b_t$

GRU state : $h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t$

# Making minGRU

Parallel scan: $v_t = a_t \odot v_{t-1} + b_t$

GRU state: $h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t$

BUT: $z_t$ and $\tilde{h}_t$ are dependent on $h_{t-1}$

$$z_t = \sigma(\text{linear}([x_t, h_{t-1}]))$$
$$\tilde{h}_t = \tanh(\text{linear}([x_t, h_{t-1}]))$$

# Making minGRU

Parallel scan: $v_t = a_t \odot v_{t-1} + b_t$

GRU state: $h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t$

BUT: $z_t$ and $\tilde{h}_t$ are <u>dependent</u> on $h_{t-1}$

$$z_t = \sigma(\text{linear}([x_t, \cancel{h_{t-1}}]))$$

$$\tilde{h}_t = \tanh(\text{linear}([x_t, \cancel{h_{t-1}}]))$$

So remove $h_{t-1}$ terms

# Making minGRU

Parallel scan: $v_t = a_t \odot v_{t-1} + b_t$

GRU state: $h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t$

BUT: $z_t$ and $\tilde{h}_t$ are <u>dependent</u> on $h_{t-1}$

$$z_t = \sigma(\text{linear}(x_t))$$

$$\tilde{h}_t = \tanh(\text{linear}(x_t))$$

# Making minGRU

Parallel scan: $v_t = a_t \odot v_{t-1} + b_t$

GRU state: $h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t$

BUT: $z_t$ and $\tilde{h}_t$ are <u>dependent</u> on $h_{t-1}$

$z_t = \sigma(\text{linear}(x_t))$

$\tilde{h}_t = \sout{\tanh}(\text{linear}(x_t))$

And remove range restriction on $\tilde{h}_t$ as not required

# Making minGRU

**GRU**

$$
\begin{aligned}
\boldsymbol{h}_t &= (\mathbf{1} - \boldsymbol{z}_t) \odot \boldsymbol{h}_{t-1} + \boldsymbol{z}_t \odot \tilde{\boldsymbol{h}}_t \\
\boldsymbol{z}_t &= \sigma(\text{Linear}_{d_h}([\boldsymbol{x}_t, \boldsymbol{h}_{t-1}])) \\
\boldsymbol{r}_t &= \sigma(\text{Linear}_{d_h}([\boldsymbol{x}_t, \boldsymbol{h}_{t-1}])) \\
\tilde{\boldsymbol{h}}_t &= \tanh(\text{Linear}_{d_h}([\boldsymbol{x}_t, \boldsymbol{r}_t \odot \boldsymbol{h}_{t-1}]))
\end{aligned}
$$

$\Rightarrow$

**minGRU**

$$
\begin{aligned}
\boldsymbol{h}_t &= (\mathbf{1} - \boldsymbol{z}_t) \odot \boldsymbol{h}_{t-1} + \boldsymbol{z}_t \odot \tilde{\boldsymbol{h}}_t \\
\boldsymbol{z}_t &= \sigma(\text{Linear}_{d_h}(\boldsymbol{x}_t)) \\
\tilde{\boldsymbol{h}}_t &= \text{Linear}_{d_h}(\boldsymbol{x}_t)
\end{aligned}
$$

Note $h_t$ scale is time independent because $z_t$ and $(1-z_t)$ sum to 1

# Making minLSTM

Same process as for GRU

**LSTM**

$$\boldsymbol{h}_t = \boldsymbol{o}_t \odot \tanh(\boldsymbol{c}_t)$$
$$\boldsymbol{o}_t = \sigma(\text{Linear}_{d_h}([\boldsymbol{x}_t, \boldsymbol{h}_{t-1}]))$$
$$\boldsymbol{c}_t = \boldsymbol{f}_t \odot \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \odot \tilde{\boldsymbol{c}}_t$$
$$\boldsymbol{f}_t = \sigma(\text{Linear}_{d_h}([\boldsymbol{x}_t, \boldsymbol{h}_{t-1}]))$$
$$\boldsymbol{i}_t = \sigma(\text{Linear}_{d_h}([\boldsymbol{x}_t, \boldsymbol{h}_{t-1}]))$$
$$\tilde{\boldsymbol{c}}_t = \tanh(\text{Linear}_{d_h}([\boldsymbol{x}_t, \boldsymbol{h}_{t-1}]))$$

$\Rightarrow$

**minLSTM**

$$\boldsymbol{h}_t = \boldsymbol{f}'_t \odot \boldsymbol{h}_{t-1} + \boldsymbol{i}'_t \odot \tilde{\boldsymbol{h}}_t$$
$$\boldsymbol{f}_t = \sigma(\text{Linear}_{d_h}(\boldsymbol{x}_t))$$
$$\boldsymbol{i}_t = \sigma(\text{Linear}_{d_h}(\boldsymbol{x}_t))$$
$$\tilde{\boldsymbol{h}}_t = \text{Linear}_{d_h}(\boldsymbol{x}_t)$$
$$\boldsymbol{f}'_t, \boldsymbol{i}'_t \leftarrow \frac{\boldsymbol{f}_t}{\boldsymbol{f}_t + \boldsymbol{i}_t}, \frac{\boldsymbol{i}_t}{\boldsymbol{f}_t + \boldsymbol{i}_t}$$

But $h_t$ scale is time dependent as $f_t$ and $i_t$ are computed independently, so normalize

# Were RNNs all we needed? Efficiency

- Significant time improvements
  - "As such, in a setting where minGRU would take a day to finish training for a fixed number of epochs, its traditional counterpart GRU could take over 3 years."
- At the cost of memory (more than originals, but better than Mamba)

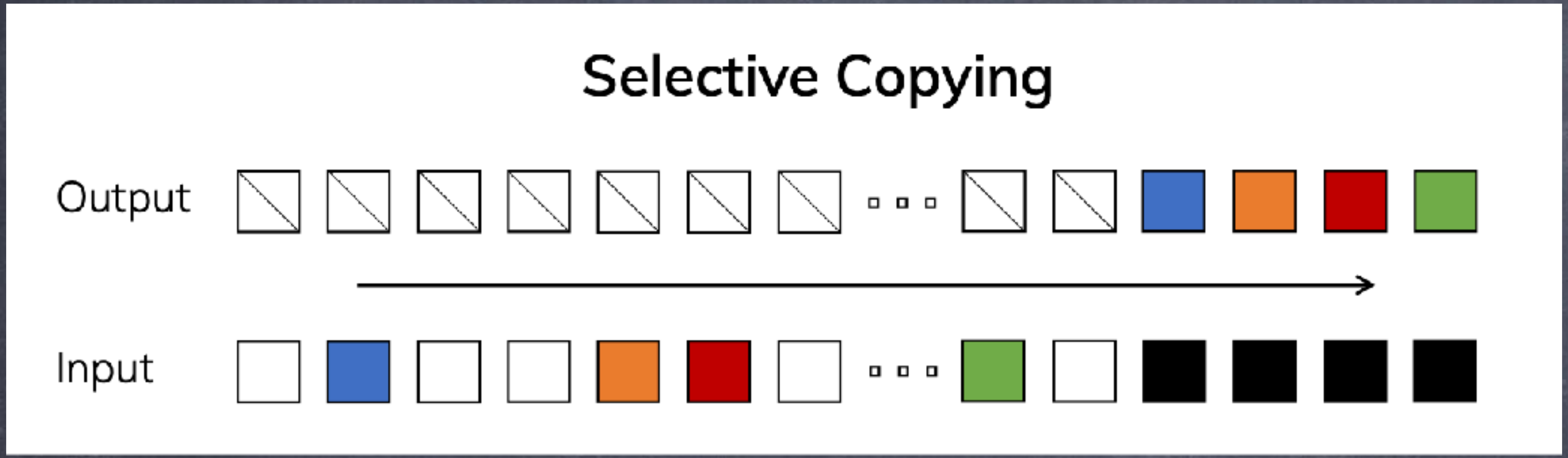
# Were RNNs all we needed?
# Effect of dropping $h_{t-1}$

- Worse performance on certain tasks by removing time dependent gates
- But can make up for this by stacking (which hasn't really had very much traction in traditional RNNs)

| Model | # Layers | Accuracy |
|---|---|---|
| MinLSTM | 1 | 37.6 ± 2.0 |
| | 2 | 85.7 ± 5.8 |
| | 3 | 96.0 ± 2.8 |
| MinGRU | 1 | 37.0 ± 2.3 |
| | 2 | 96.8 ± 3.2 |
| | 3 | 99.5 ± 0.2 |

Table 1: Comparison of the number of layers on the Selective Copying Task (Gu & Dao, 2024).

# Were RNNs all we needed?
# Performance - copying




The Selective Copying task has random spacing in between inputs and requires time-varying models that can selectively remember or ignore inputs depending on their content.

| Model | Layer | Accuracy |
|---|---|---|
| H3 | Hyena | 30.1 |
| Mamba | Hyena | 28.4 |
| S4 | S4 | 18.3 |
| H3 | S4 | 57.0 |
| Mamba | S4 | 56.4 |
| S4 | S6 | 97.0 |
| H3 | S6 | 99.7 |
| Mamba | S6 | 99.8 |
| minGRU | minGRU | 99.5 ± 0.2 |
| minLSTM | minLSTM | 96.0 ± 2.8 |

Table 2: Selective Copy Task. minLSTM, minGRU, and Mamba's S6 (Gu & Dao, 2024) are capable of solving this task. Other methods such as S4, H3, and Hyena at best only partially solve the task.

# Were RNNs all we needed?
# Performance - RL

| Dataset | DT | DS4 | DAaren | DMamba | minLSTM | minGRU |
|---|---|---|---|---|---|---|
| HalfCheetah-M | 42.6 | 42.5 | 42.2 | 42.8 | 42.7 ± 0.7 | 43.0 ± 0.4 |
| Hopper-M | 68.4 | 54.2 | 80.9 | 83.5 | 85.0 ± 4.4 | 79.4 ± 8.2 |
| Walker-M | 75.5 | 78.0 | 74.4 | 78.2 | 72.0 ± 7.5 | 73.3 ± 3.3 |
| HalfCheetah-M-R | 37.0 | 15.2 | 37.9 | 39.6 | 38.6 ± 1.1 | 38.5 ± 1.1 |
| Hopper-M-R | 85.6 | 49.6 | 77.9 | 82.6 | 88.5 ± 4.7 | 90.5 ± 0.9 |
| Walker-M-R | 71.2 | 69.0 | 71.4 | 70.9 | 69.7 ± 10.7 | 72.8 ± 8.9 |
| HalfCheetah-M-E | 88.8 | 92.7 | 75.7 | 91.9 | 85.4 ± 1.7 | 86.3 ± 0.5 |
| Hopper-M-E | 109.6 | 110.8 | 103.9 | 111.1 | 110.3 ± 1.6 | 109.7 ± 2.7 |
| Walker-M-E | 109.3 | 105.7 | 110.5 | 108.3 | 110.3 ± 0.5 | 110.3 ± 0.4 |
| Average | 76.4 | 68.6 | 75.0 | 78.8 | 78.1 | 78.2 |

Table 3: Reinforcement Learning results on the D4RL (Fu et al., 2020) datasets. We report the expert normalized returns (higher is better), following (Fu et al., 2020), averaged across five random seeds. The minimal versions of LSTM and GRU, minLSTM and minGRU outperform Decision S4 (David et al., 2023) and perform comparably with Decision Mamba (Ota, 2024), (Decision) Aaren (Feng et al., 2024) and Decision Transformer (Chen et al., 2021).

# Were RNNs all we needed?
# Performance - language

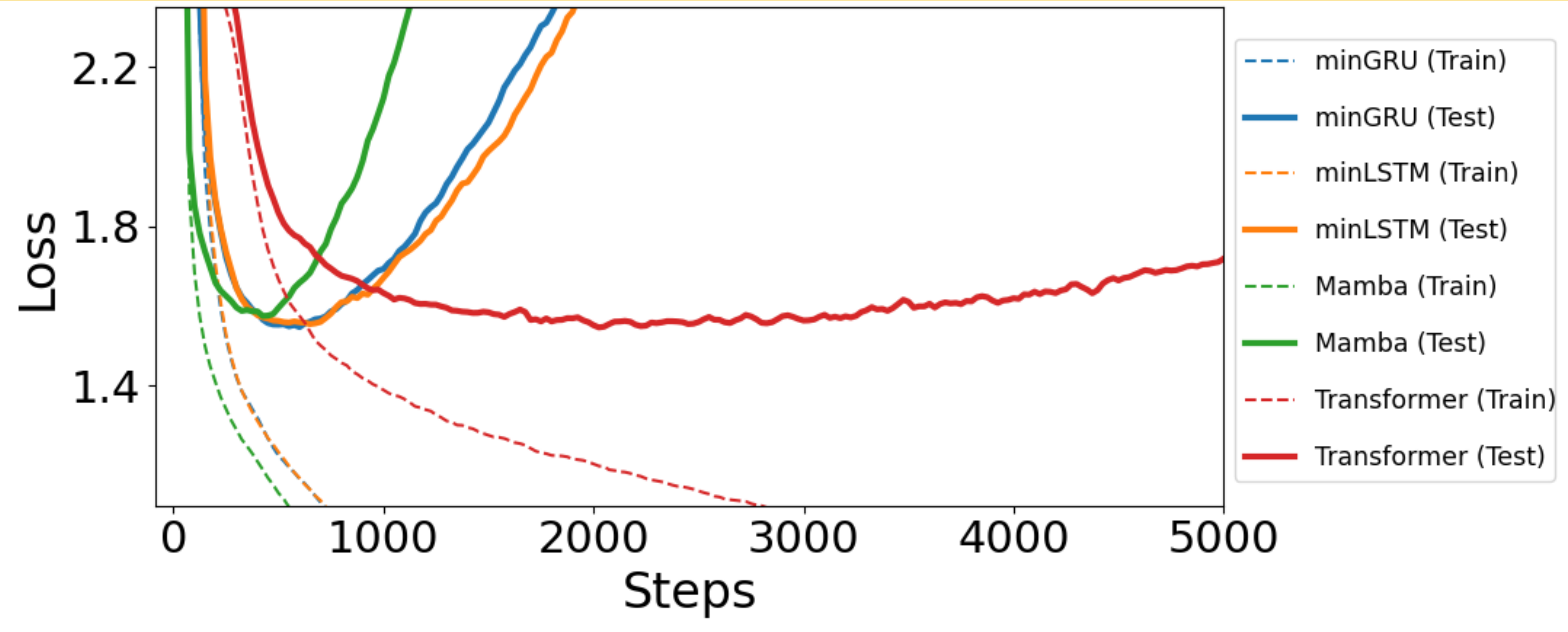



Figure 2: Language Modelling results on the Shakespeare dataset. Minimal versions of decade-old RNNs (LSTMs and GRUs) performed comparably to Mamba and Transformers. Transformers required $\sim 2.5\times$ more training steps to achieve comparable performance, overfitting eventually.

# Reviews

# Contributions

Strengths

Weaknesses

# Actual reviews

https://openreview.net/forum?id=GrmFFxGnOR

- Public comments about similarity to other work
- Public concern over the title?
- Public comments asking reviewer to re-evaluate!

- Scores: 8, 6, 3, 3

# Actual reviews

https://openreview.net/forum?id=GrmFFxGnOR

- Insufficient comparison
- Limited datasets
- Lack of depth to literature review
- "Similarities" to other models (although…)
- Lack of novelty