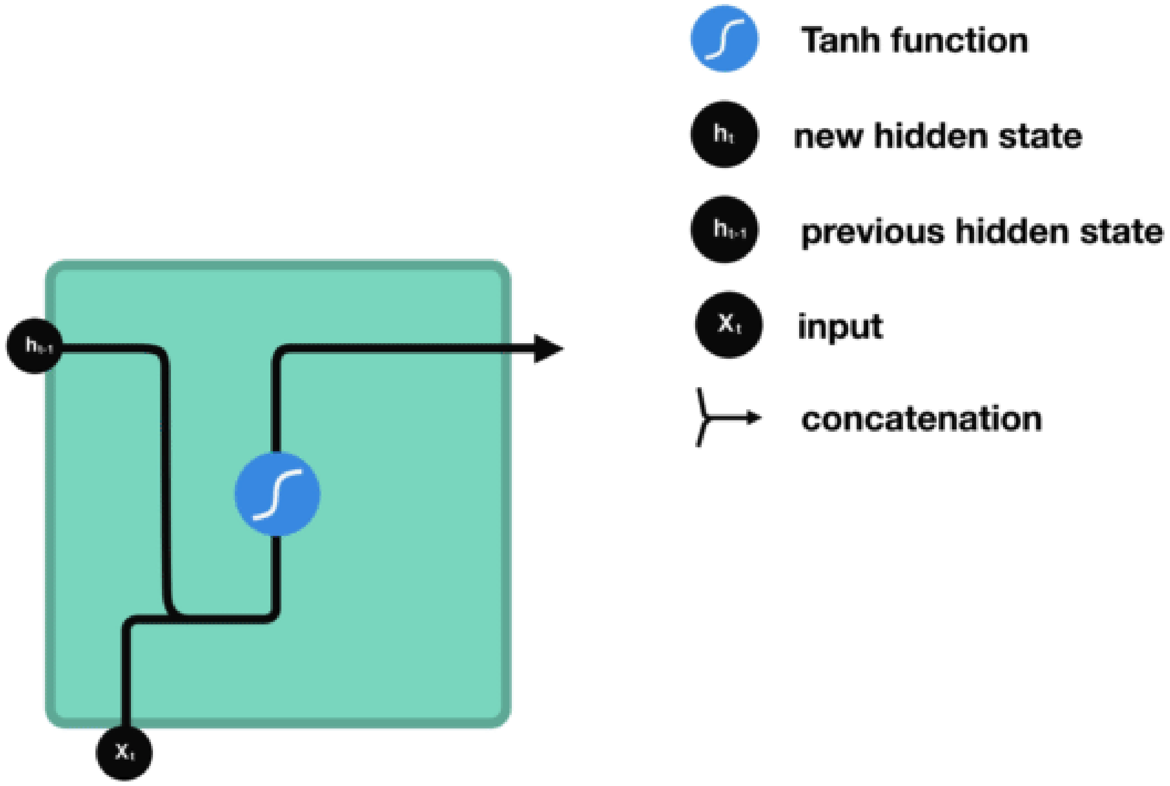
LECTURE 5:

ADVANCED RECURRENT NEURAL NETWORKS

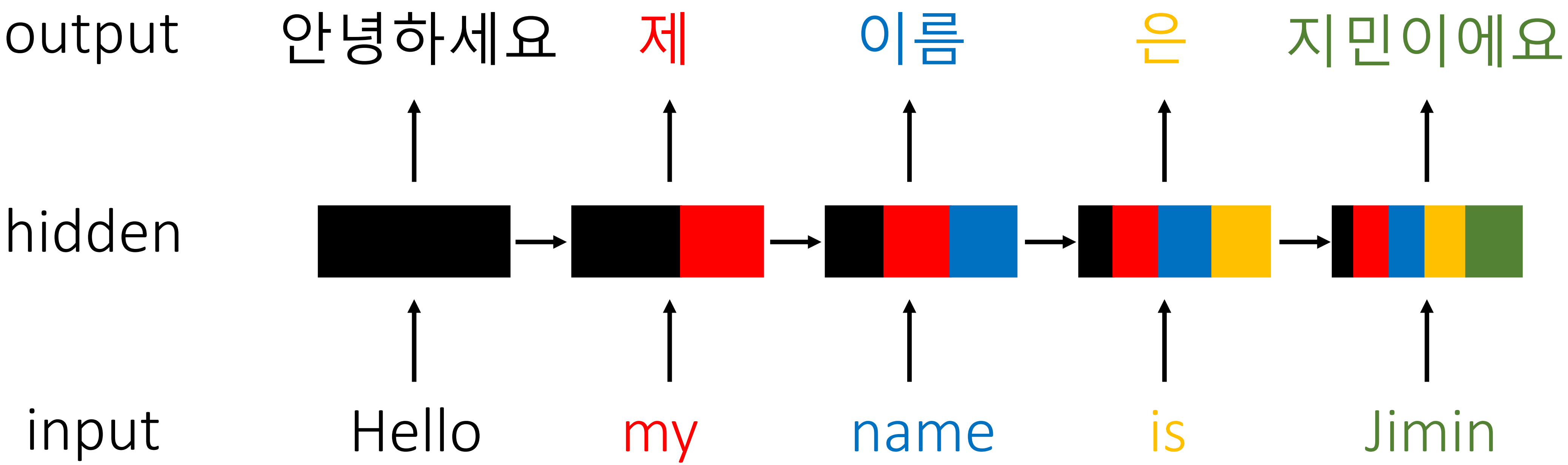
University of Washington, Seattle

Fall 2025

Previously in EEP 596…



Previously in EEP 596...



OUTLINE

Part 1: Gated RNNs

- Need for Gated RNNs
- Long Short-Term Memory (LSTM)
- Gated Recurrent Unit (GRU)
- RNN extensions on LSTM/GRU

Part 2: Encoder-Decoder RNNs

- Many to many RNN Recap
- Encoder-Decoder Architecture
- Training Encoder-Decoder RNNs

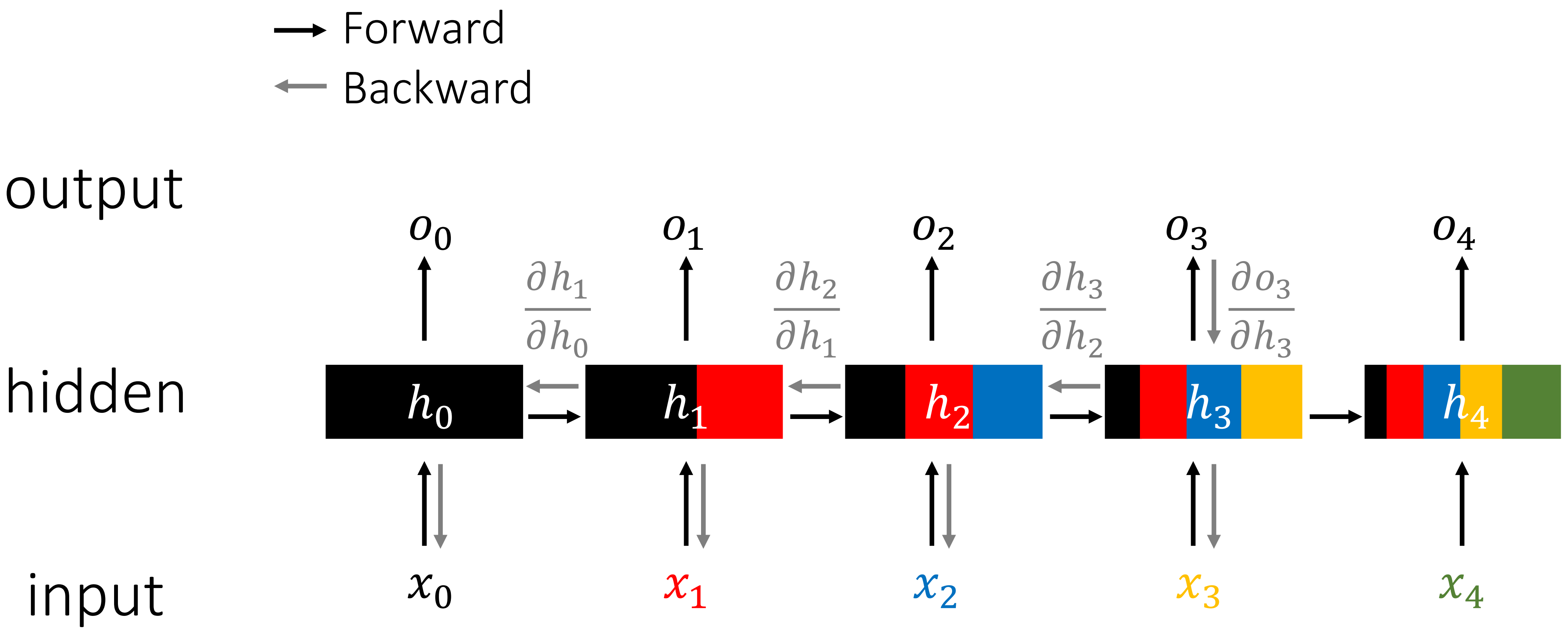
GATED RNNs

Need for Gated RNNs

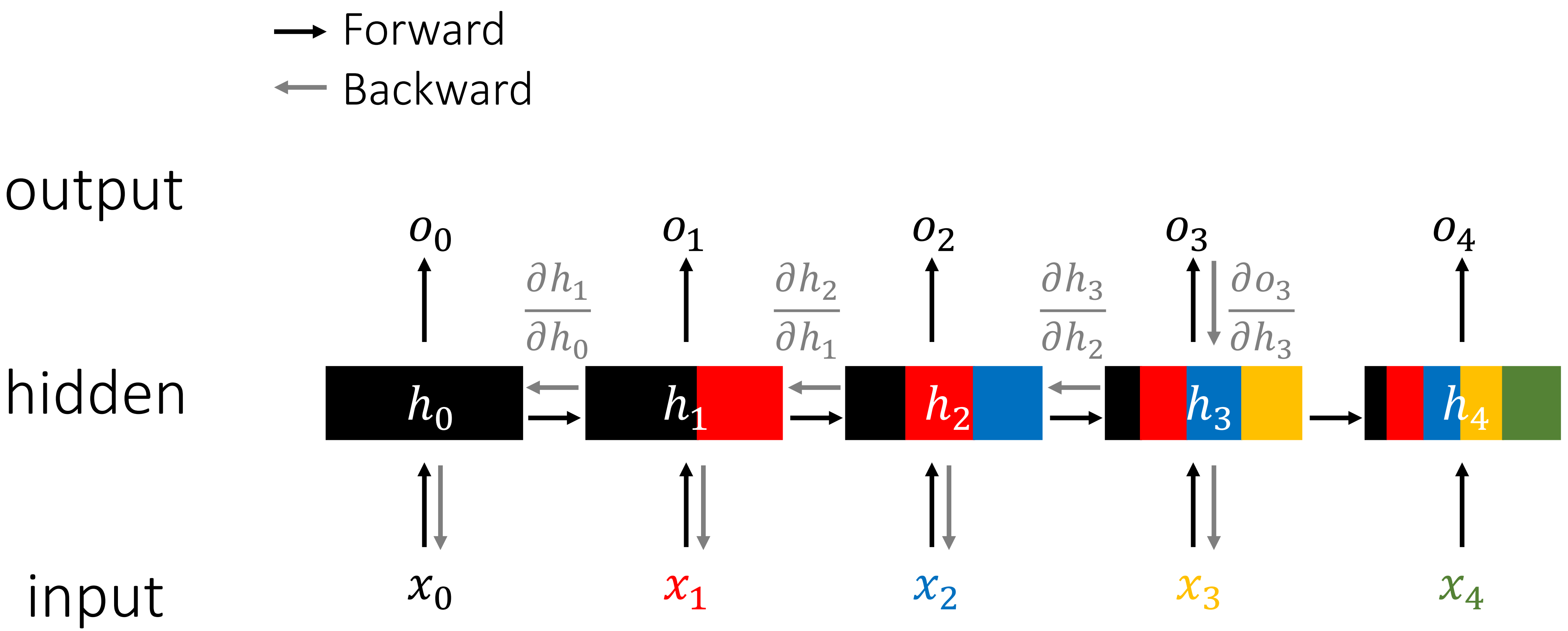
Long Short-Term Memory (LSTM)

Gated Recurrent Unit (GRU)

Recap: Backpropagation in RNNs

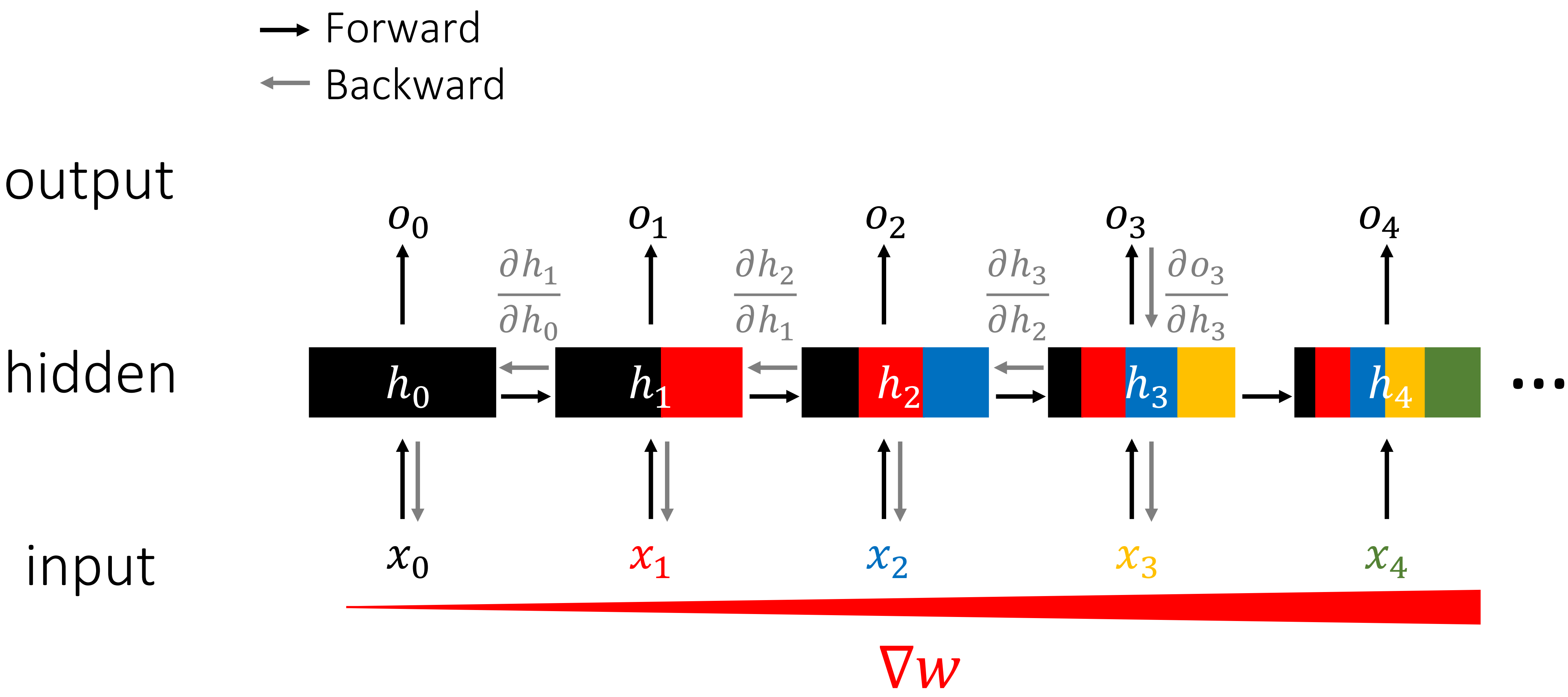


Recap: Backpropagation in RNNs

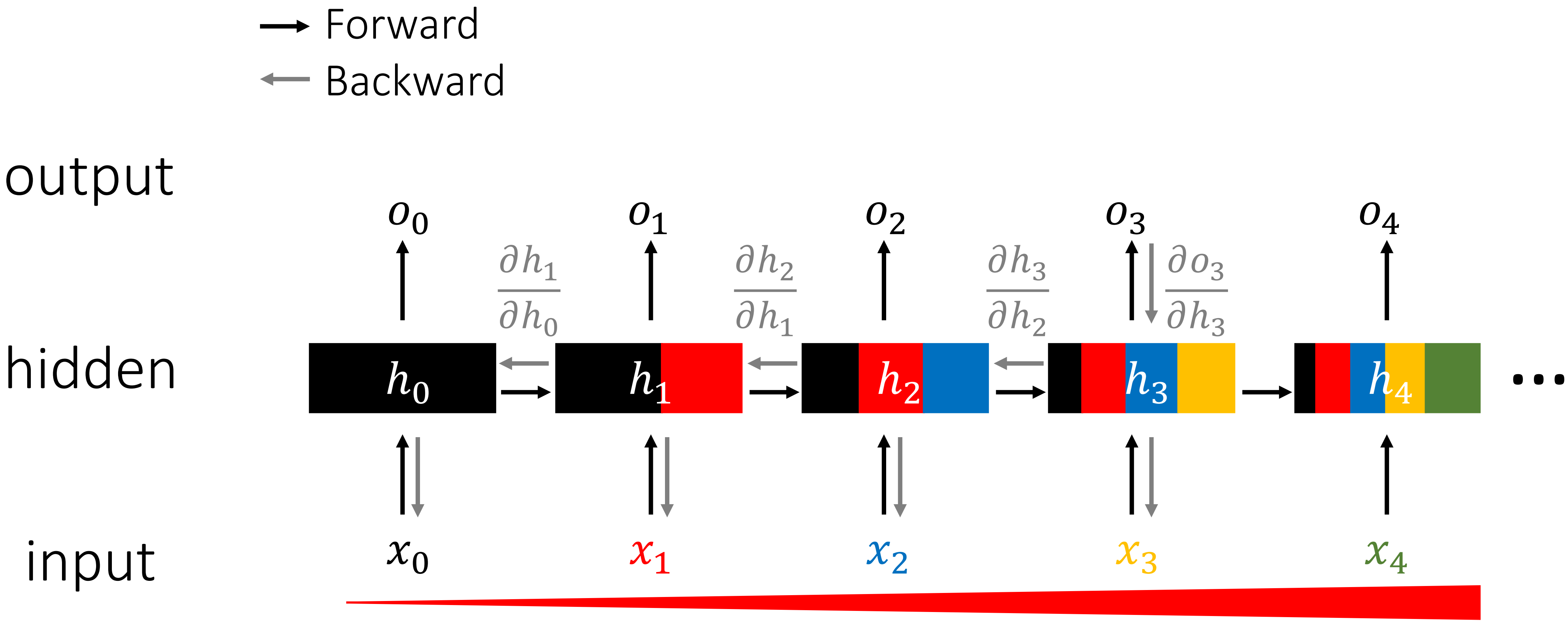


Backpropagation is performed backward in time

Vanishing and Exploding Gradients



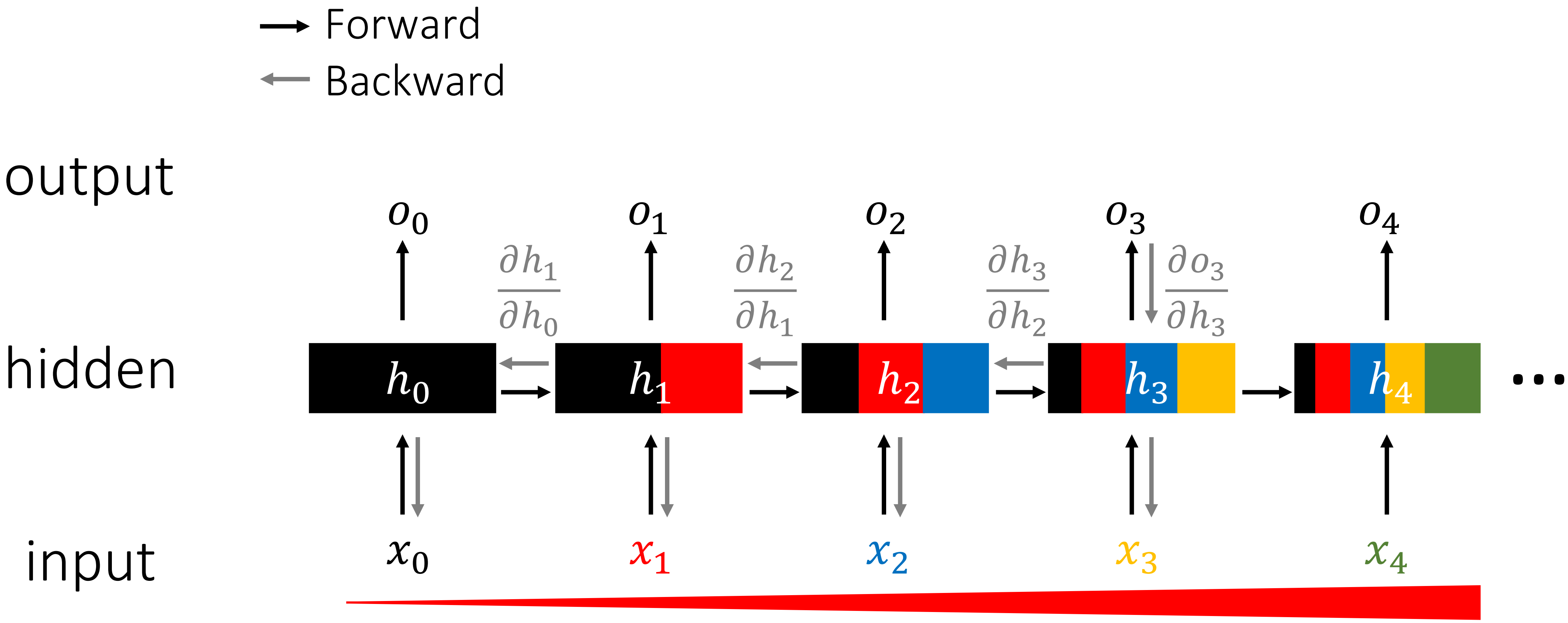
Vanishing and Exploding Gradients



Longer input sequence →

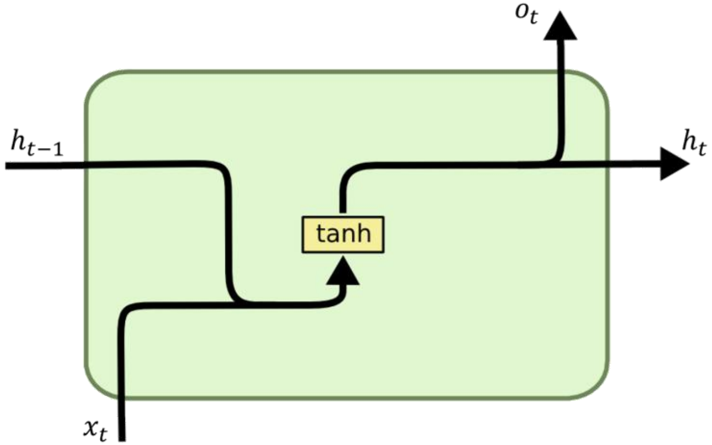
higher risk of Vanishing/Exploding Gradients!

Vanishing and Exploding Gradients

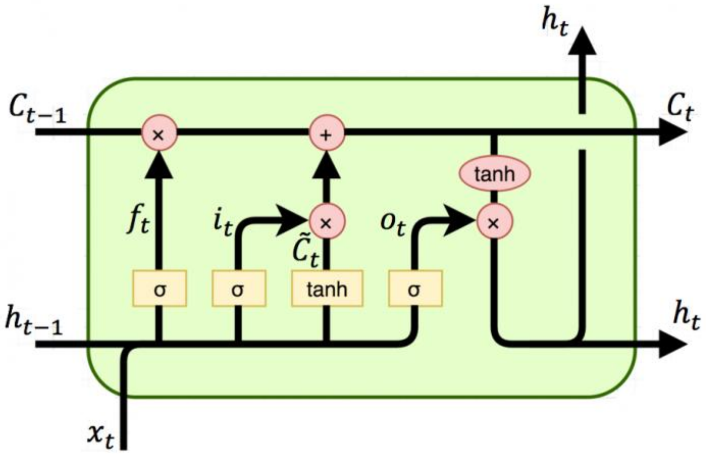


Need for better RNN architecture capable of processing longer sequence

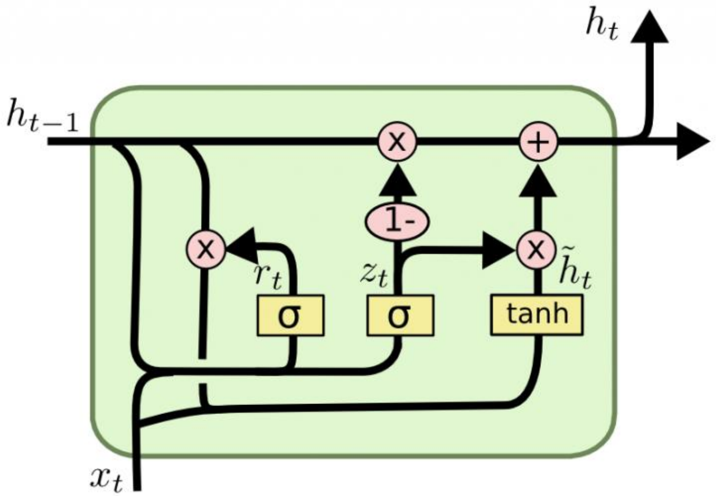
Gated RNNs



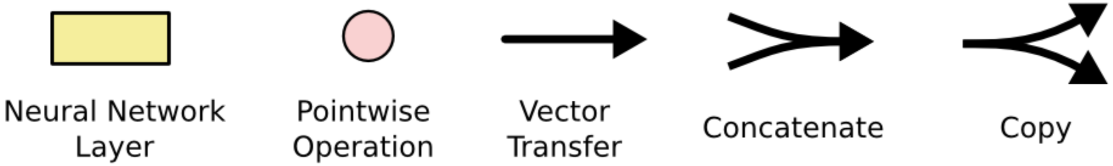
Vanilla RNN



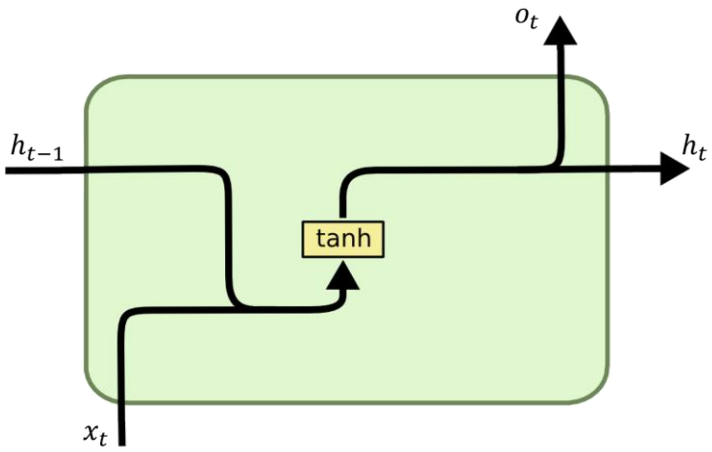
LSTM



GRU

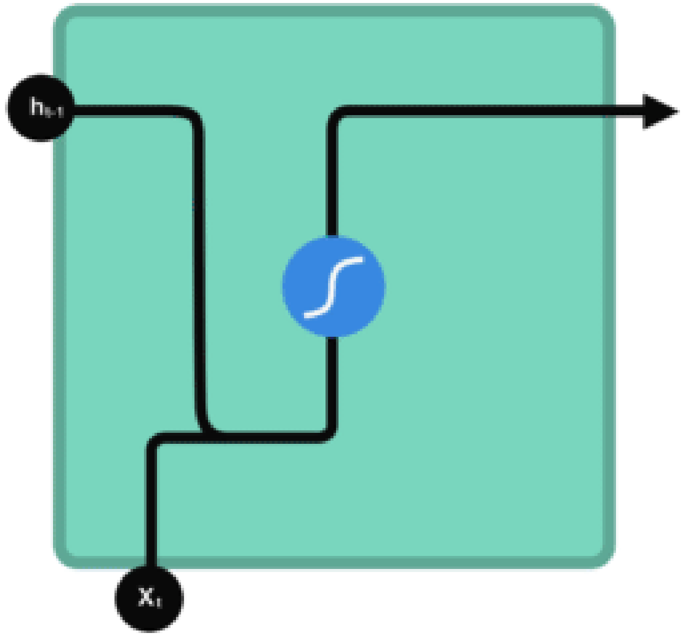


Vanilla RNN



Vanilla RNN

Vanilla RNN



Tanh function

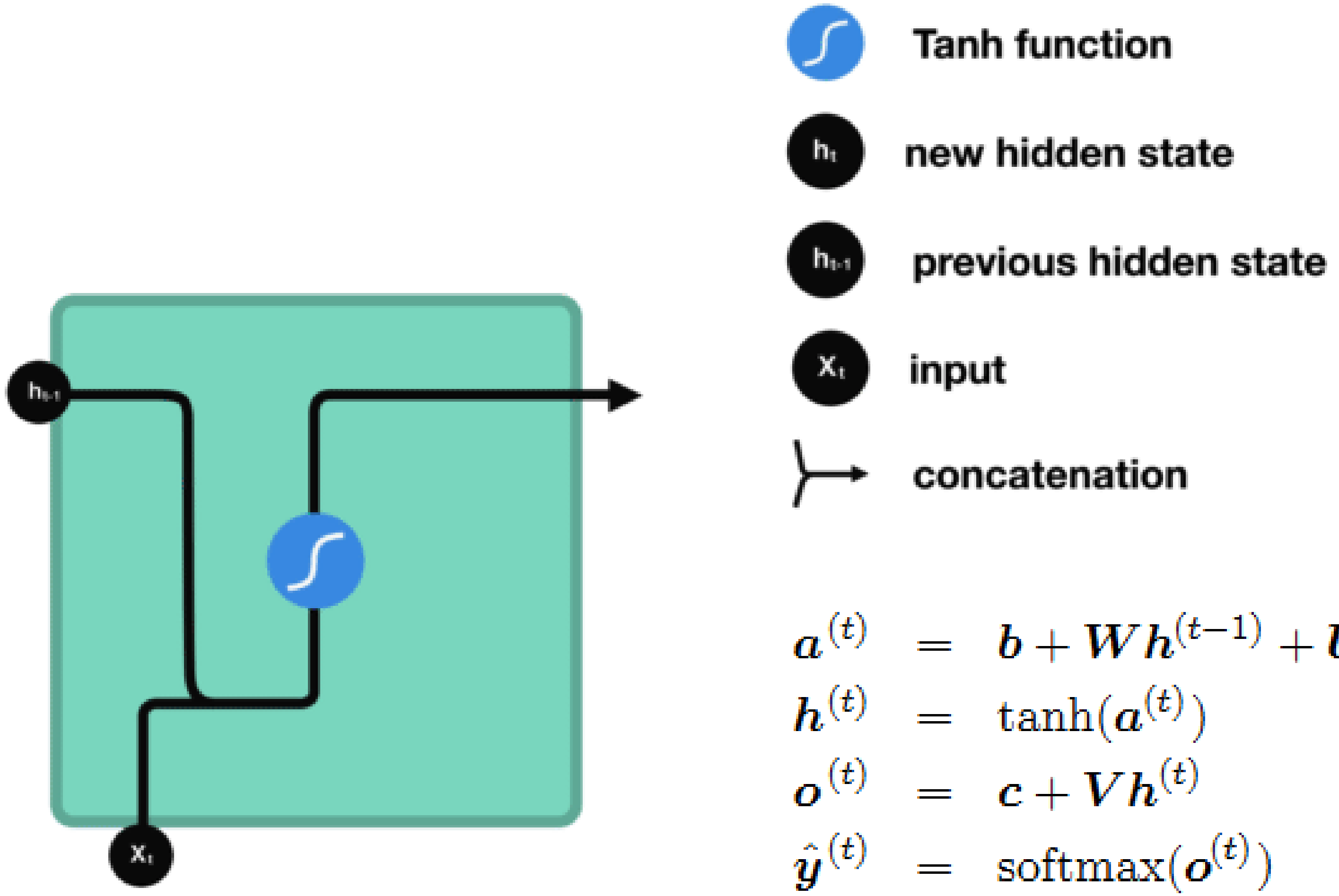
new hidden state

previous hidden state

input

concatenation

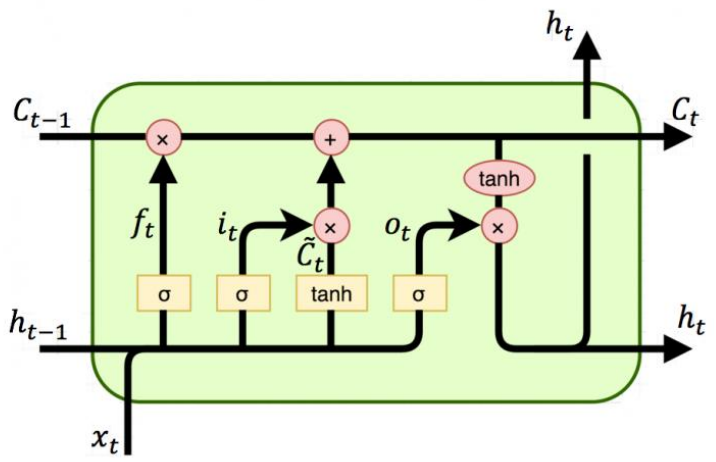
Vanilla RNN



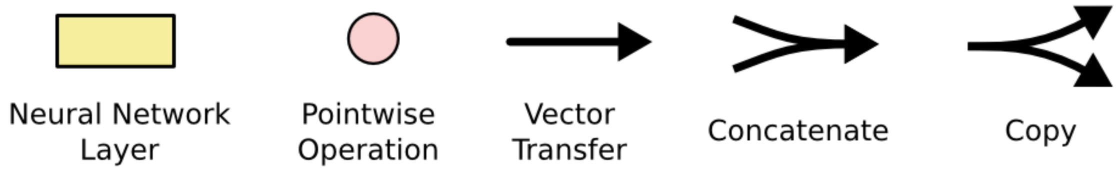
$$
\begin{aligned}
\boldsymbol{a}^{(t)} &= \boldsymbol{b} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)} \\
\boldsymbol{h}^{(t)} &= \tanh(\boldsymbol{a}^{(t)}) \\
\boldsymbol{o}^{(t)} &= \boldsymbol{c} + \boldsymbol{V}\boldsymbol{h}^{(t)} \\
\hat{\boldsymbol{y}}^{(t)} &= \operatorname{softmax}(\boldsymbol{o}^{(t)})
\end{aligned}
$$

LSTM (Long Short-Term Memory)

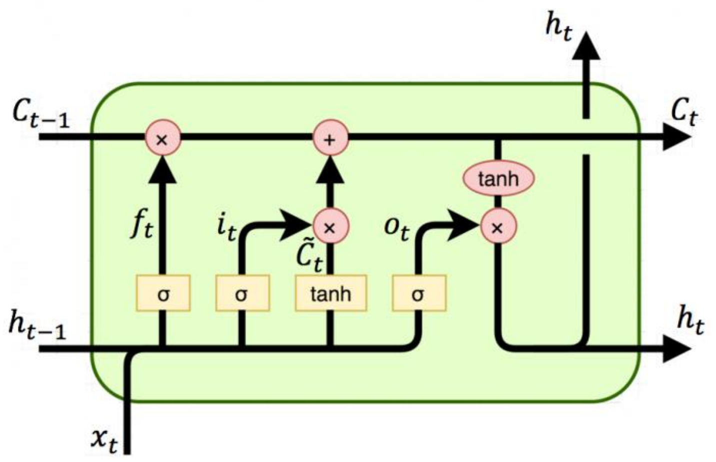
Hochreiter, Sepp, and Jürgen Schmidhuber. "Long short-term memory." *Neural computation* 9.8 (1997): 1735-1780.



LSTM

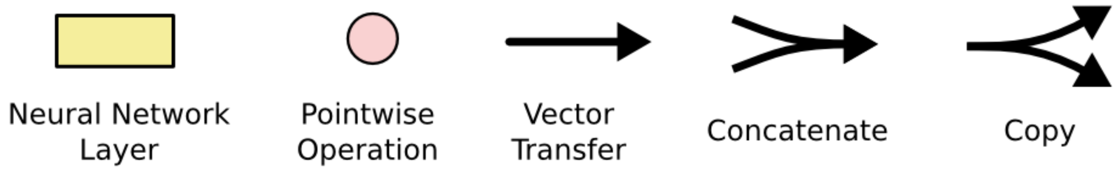


LSTM (Long Short-Term Memory)

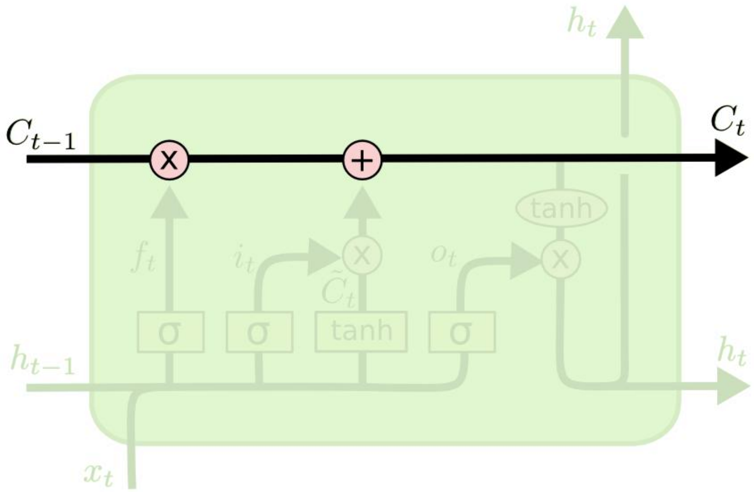


LSTM

$$
\begin{aligned}
f_t &= \sigma_g(W_f x_t + U_f h_{t-1} + b_f) \\
i_t &= \sigma_g(W_i x_t + U_i h_{t-1} + b_i) \\
o_t &= \sigma_g(W_o x_t + U_o h_{t-1} + b_o) \\
c_t &= f_t \circ c_{t-1} + i_t \circ \sigma_c(W_c x_t + U_c h_{t-1} + b_c) \\
h_t &= o_t \circ \sigma_h(c_t)
\end{aligned}
$$



LSTM: Detailed Architecture

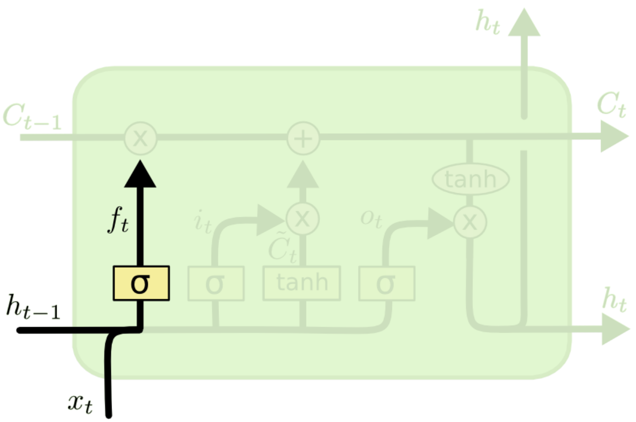


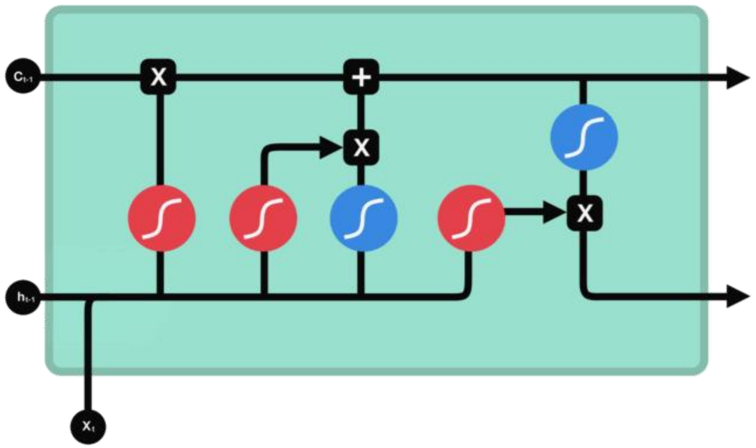
Cell state

- Unique to LSTM
- Long term memory of the model

LSTM: Detailed Architecture

$$f_t = \sigma_g(W_f x_t + U_f h_{t-1} + b_f)$$

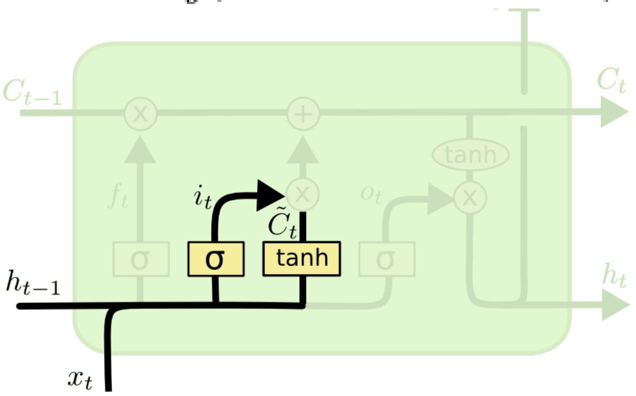


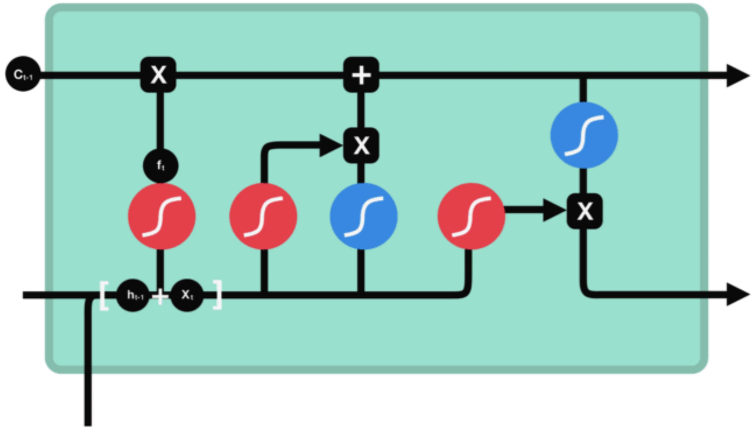


previous cell state
forget gate output

Forget gate layer
(information to forget)

$$i_t = \sigma_g(W_i x_t + U_i h_{t-1} + b_i)$$



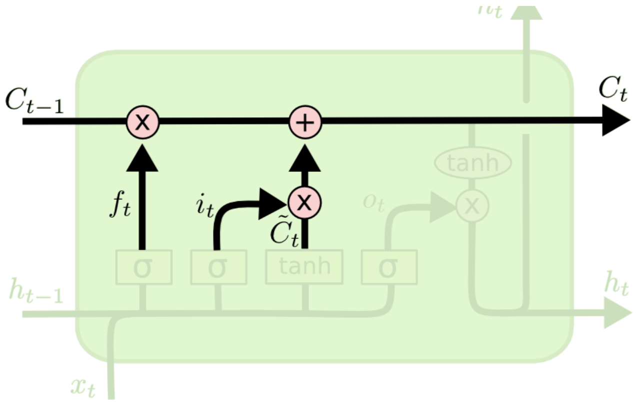


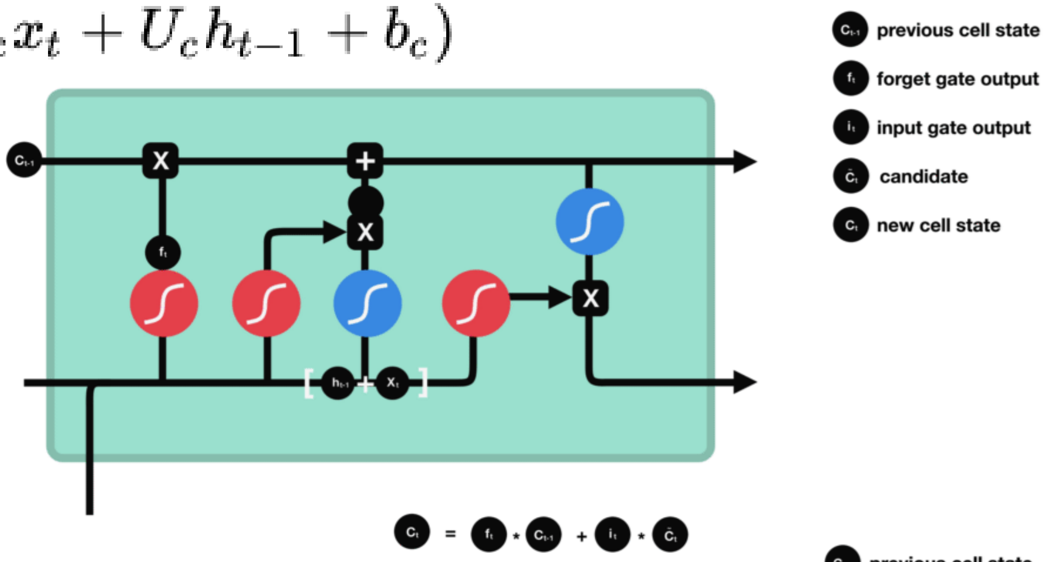
previous cell state
forget gate output
input gate output
candidate

Input gate layer
(information to keep)

LSTM: Detailed Architecture

$$c_t = f_t \circ c_{t-1} + i_t \circ \sigma_c(W_c x_t + U_c h_{t-1} + b_c)$$

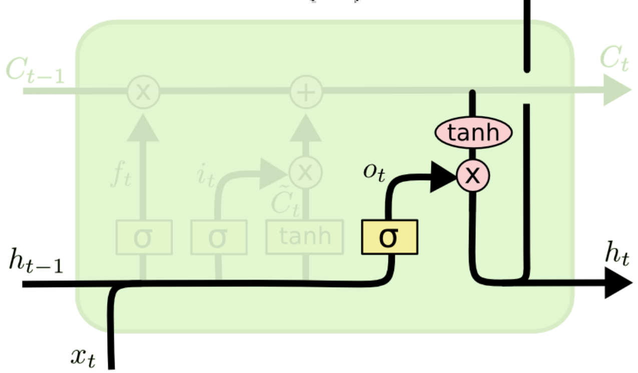


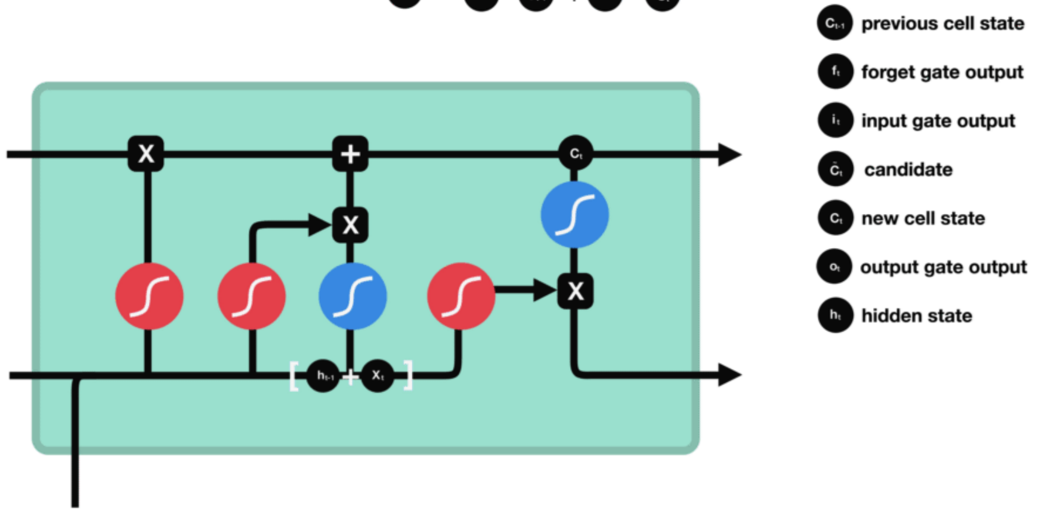


Update cell state
(Forget + Add new info)

$$o_t = \sigma_g(W_o x_t + U_o h_{t-1} + b_o)$$

$$h_t = o_t \circ \sigma_h(c_t)$$





Output gate layer
(Form new hidden state)

LSTM: Detailed Architecture

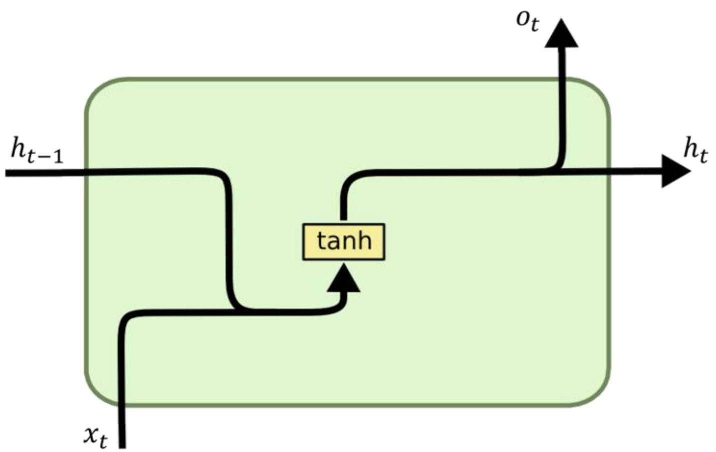
Forget gate f_t
Decides what is relevant to keep from previous steps

Input gate i_t
Decides what information is relevant to add from the current step

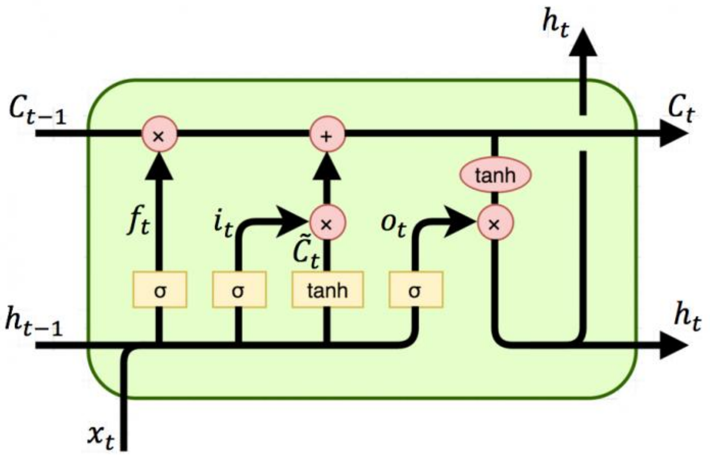
Output Gate o_t
Combined with cell state to determines next hidden state

LSTM divides **original hidden state** into
1. **long-term memory (cell state)** and 2. **context (LSTM hidden state)**

LSTM (Long Short-Term Memory)



Vanilla RNN



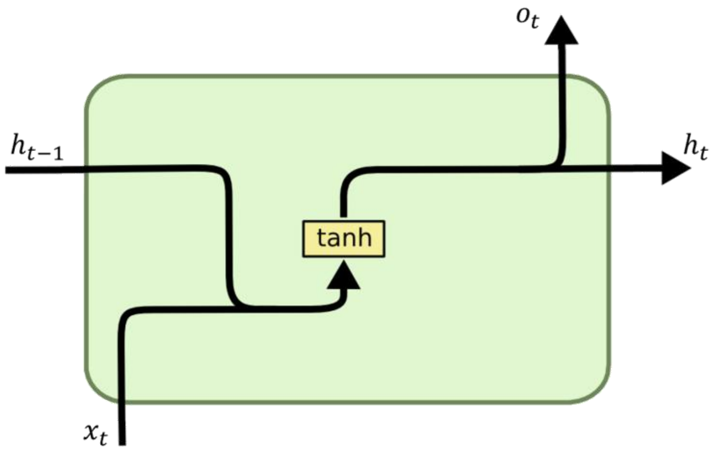
LSTM

$$h_t = \sigma(wh_{t-1}).$$

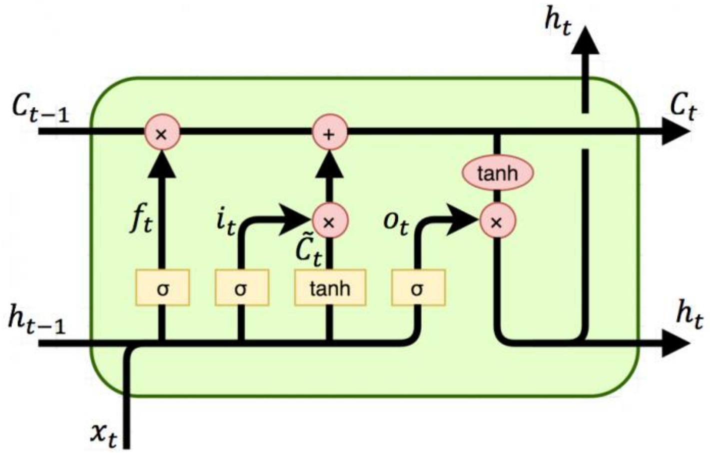
$$\frac{\partial h_{t'}}{\partial h_t} = \prod_{k=1}^{t'-t} w\sigma'(wh_{t'-k})$$
$$= \underbrace{w^{t'-t}}_{!!!} \prod_{k=1}^{t'-t} \sigma'(wh_{t'-k})$$

$$\frac{\partial c_{t'}}{\partial c_t} = \prod_{k=1}^{t'-t} \sigma(v_{t+k}).$$

LSTM (Long Short-Term Memory)



Vanilla RNN



LSTM

$$h_t = \sigma(w h_{t-1}).$$

$$\frac{\partial h_{t'}}{\partial h_t} = \prod_{k=1}^{t'-t} w\sigma'(w h_{t'-k})$$

$$= \underbrace{w^{t'-t}}_{!!!} \prod_{k=1}^{t'-t} \sigma'(w h_{t'-k})$$

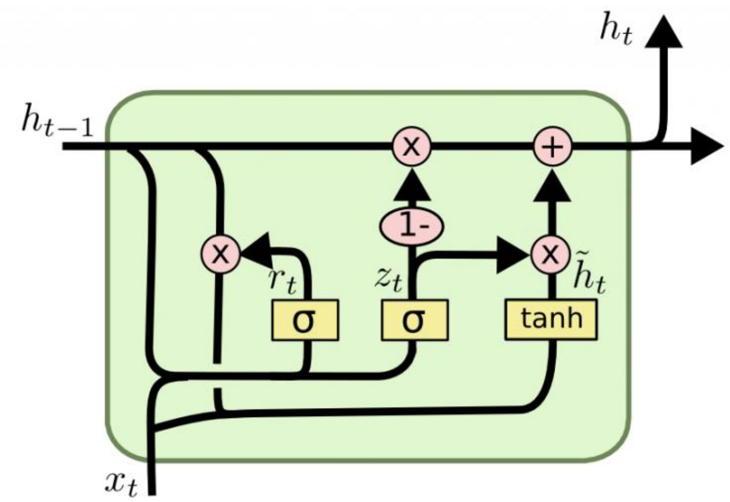
$$\frac{\partial c_{t'}}{\partial c_t} = \prod_{k=1}^{t'-t} \sigma(v_{t+k}).$$

Gradient decays or grow exponentially if $w \neq 1$

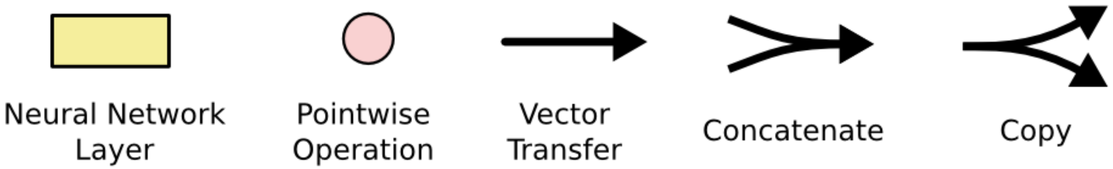
No exponential decay or growth term

Gated RNNs

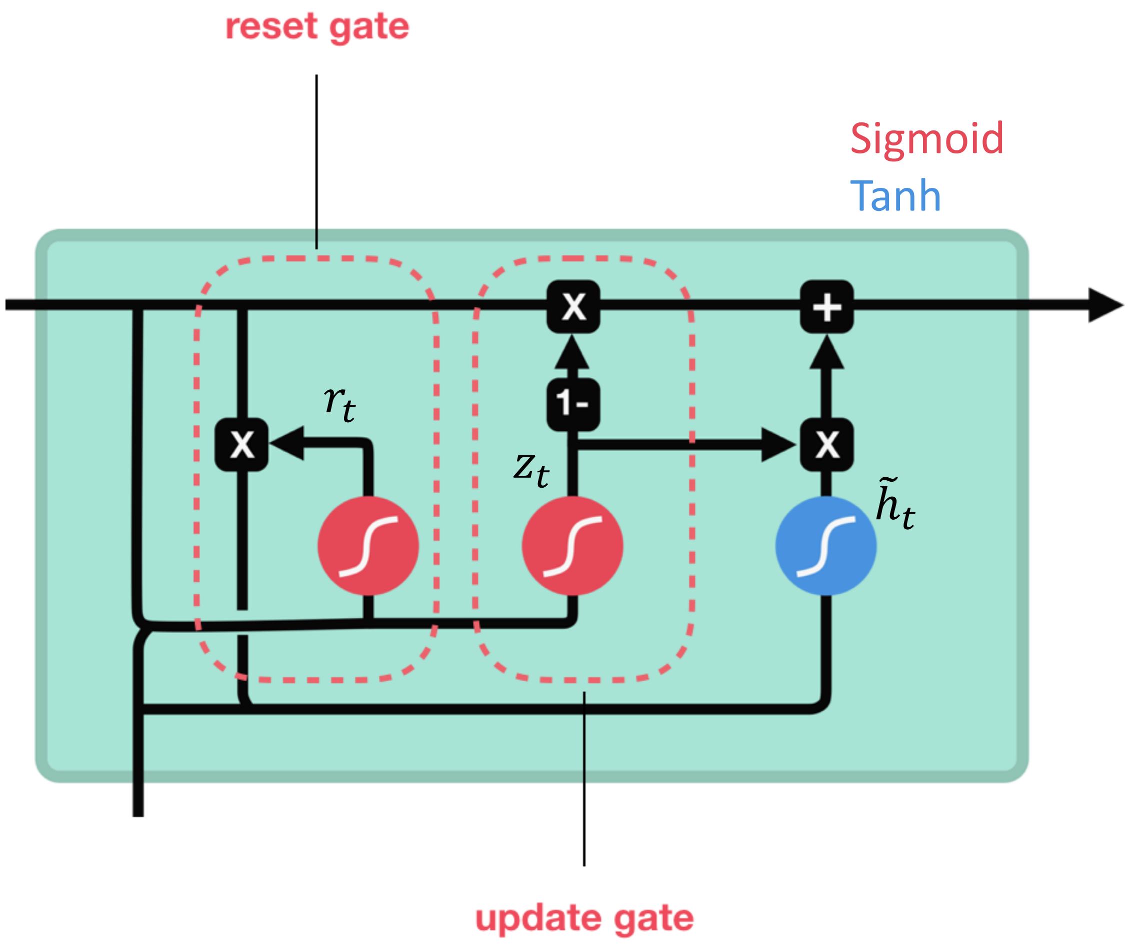
Chung, Junyoung, et al. "Empirical evaluation of gated recurrent neural networks on sequence modeling." *arXiv preprint arXiv:1412.3555* (2014).



GRU



GRU: Detailed Architecture



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right) \quad \in [0,1]$$

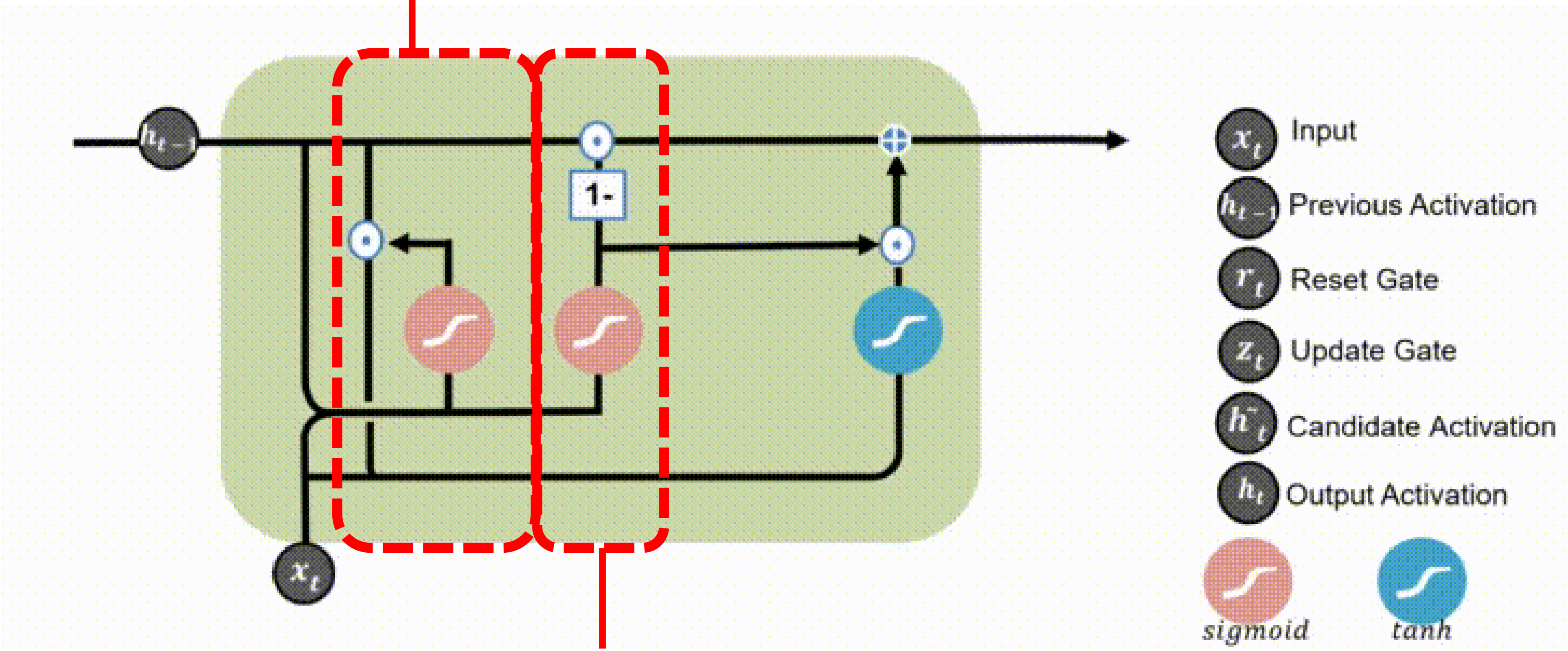
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right) \quad \in [0,1]$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right) \quad \in [-1,1]$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Information Flow in GRU

Reset Gate



Update Gate

Information Flow in GRU

Reset gate r_t
How much of the past information h_{t-1} should be retained with respect to new input x_t to form new $\tilde{h}_t$ candidate

Update gate z_t
$(1 - z_t)$ How much of the past information h_{t-1} should be discarded
(z_t) How much of new information $\tilde{h}_t$ should make into final h_t

GRU keeps original RNN input-output structure (x_t, h_t) by letting z_t to handle both data **retention** and **attrition**

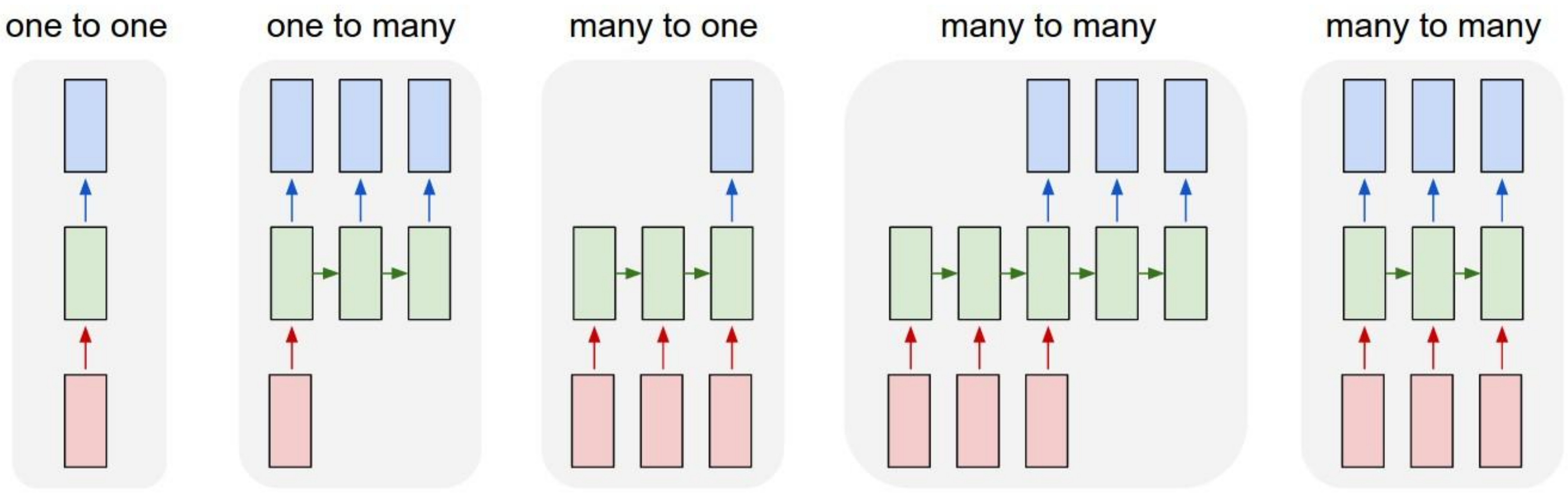
ENCODER-DECODER RNNs

Many-to-Many RNN Recap

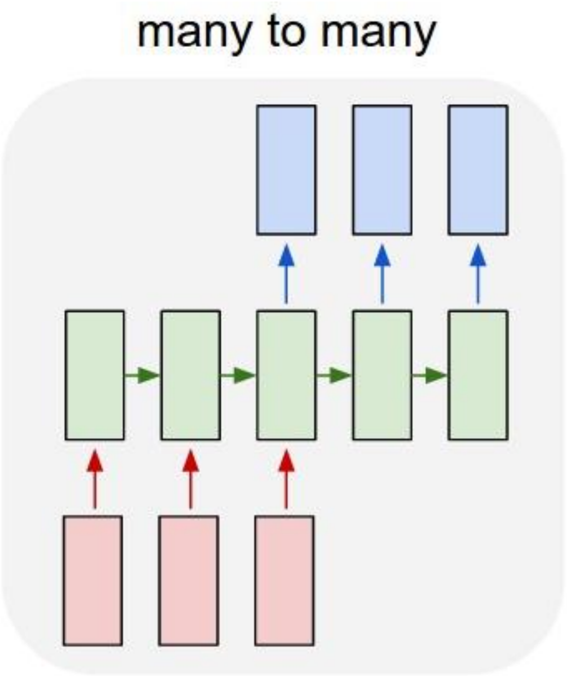
Encoder-Decoder Architecture

Training Encoder-Decoder RNNs

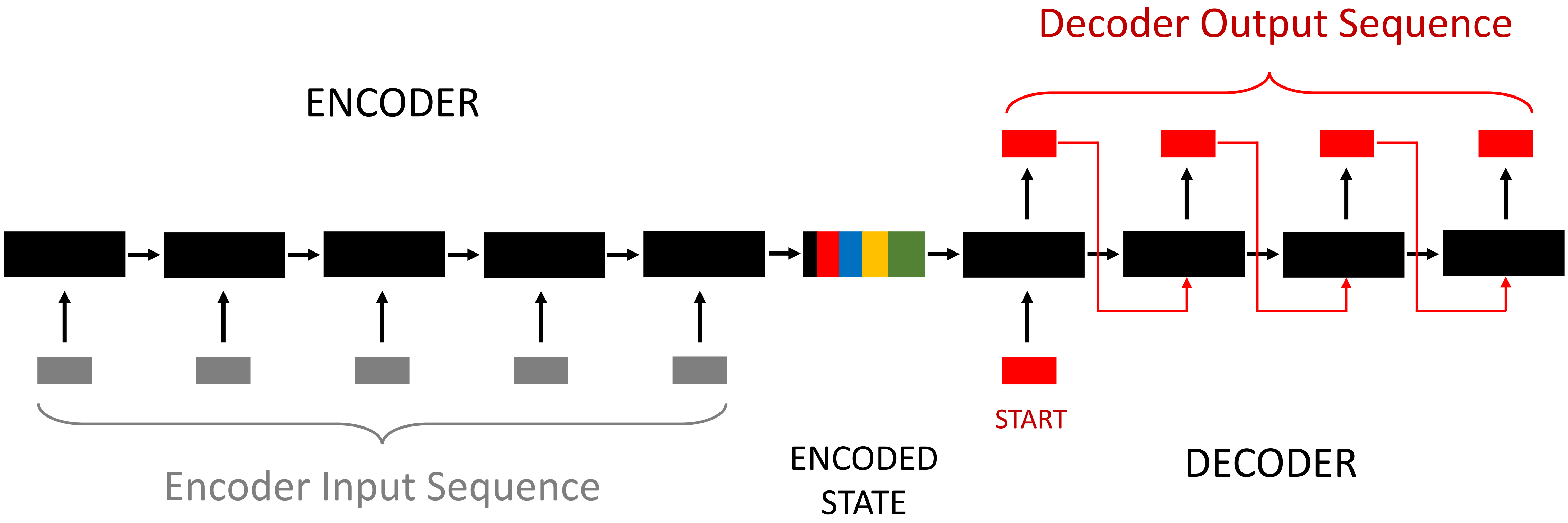
RNN Configurations



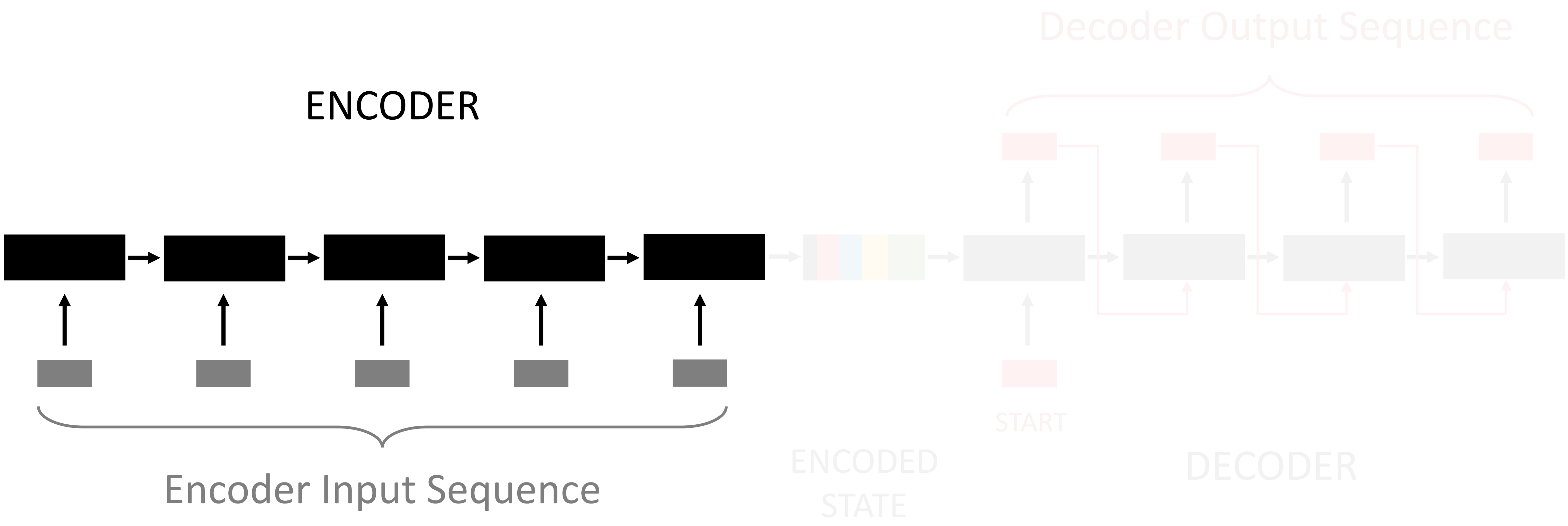
Many-to-Many



Encoder-Decoder Architecture

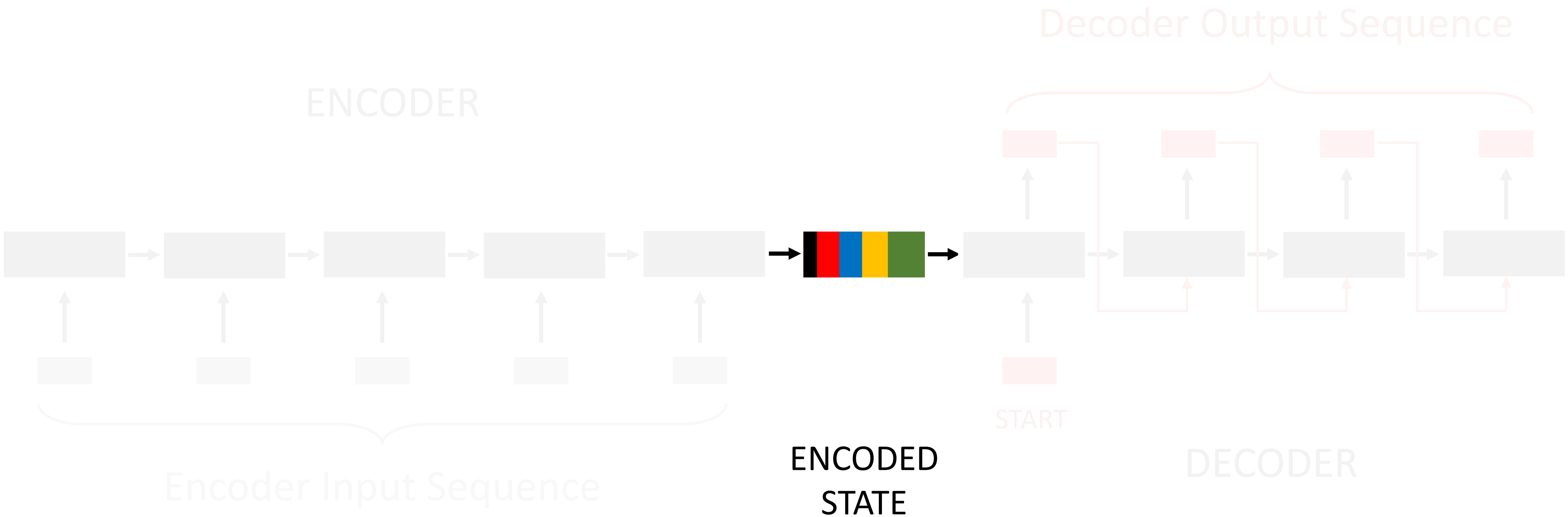


Encoder

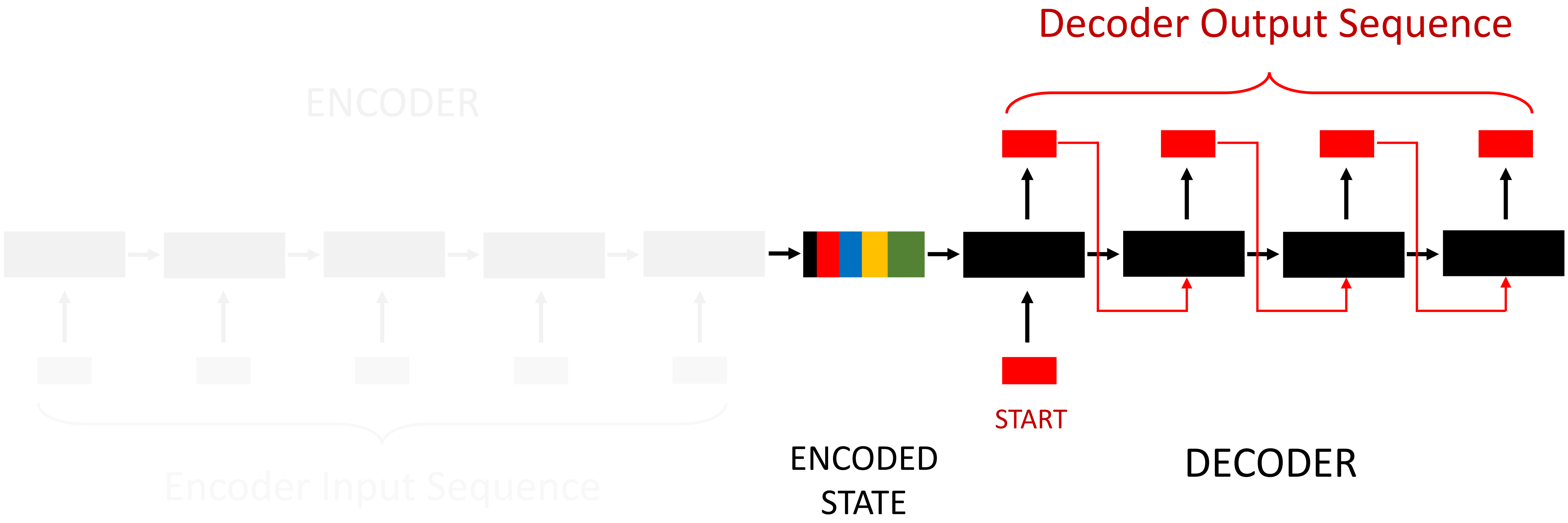


ENCODER
Encoder Input Sequence
Decoder Output Sequence
START
ENCODED
STATE
DECODER

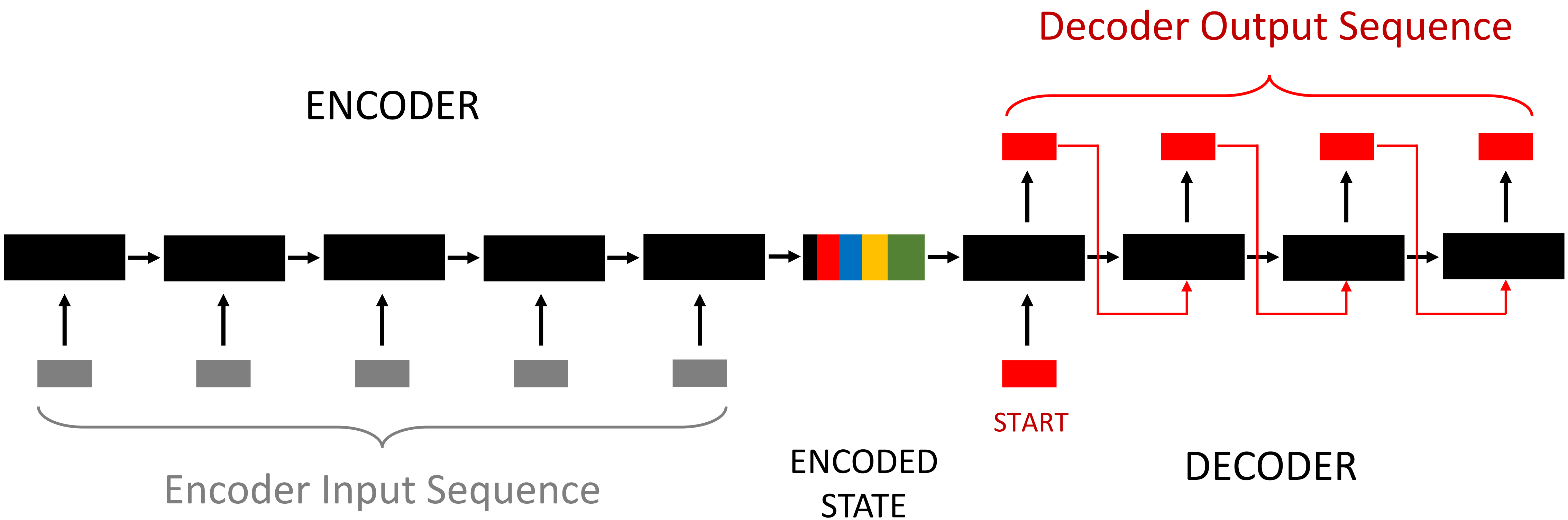
Encoded State



Decoder


Decoder Output Sequence
ENCODER
START
ENCODED
STATE
DECODER
Encoder Input Sequence

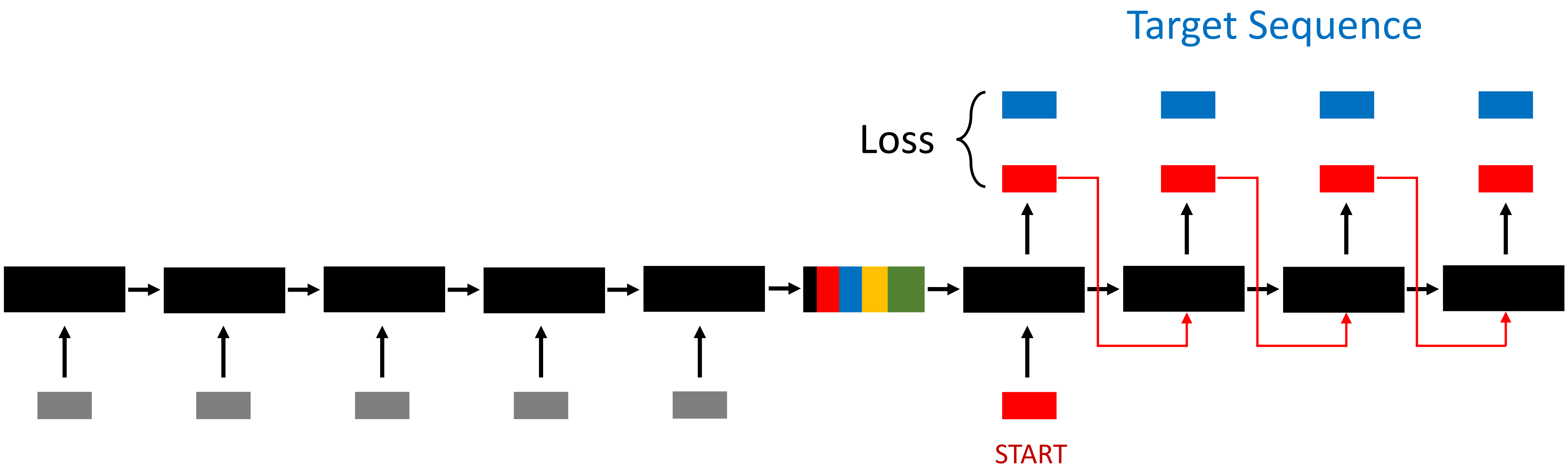
Encoder-Decoder Architecture



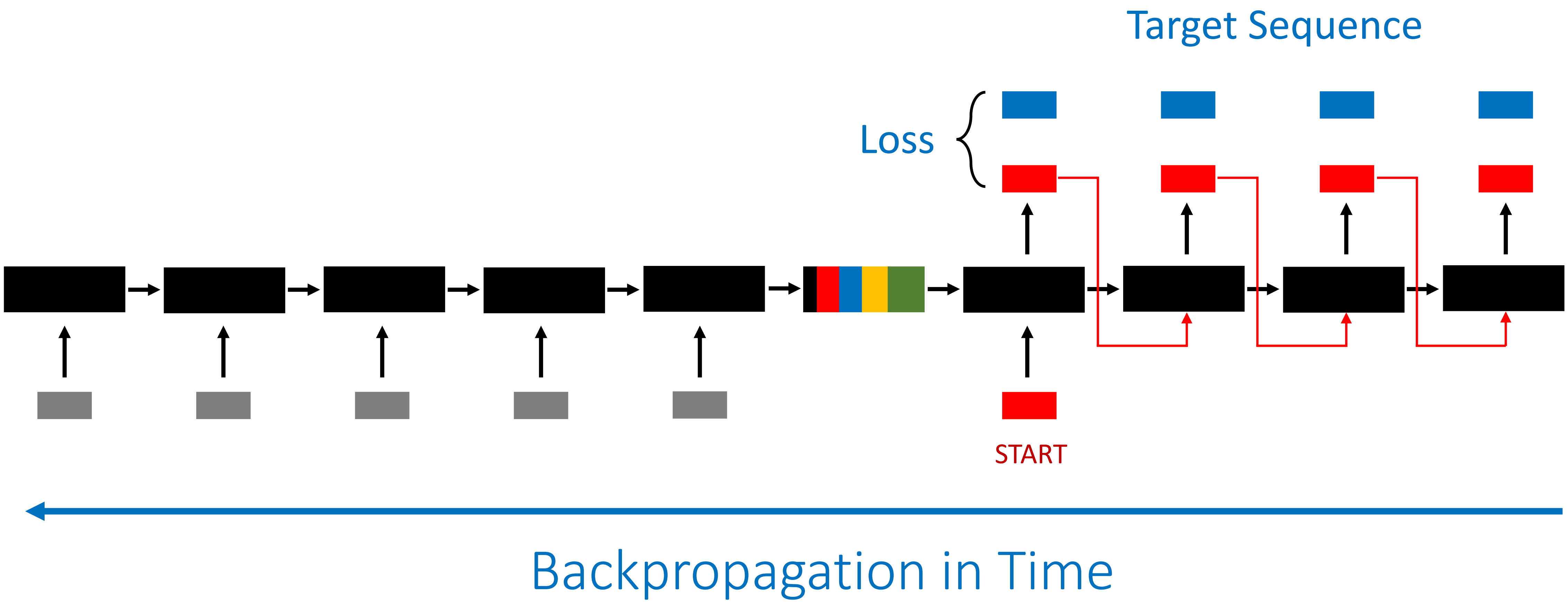
Input sequence length to Encoder (Tx) can be different from the output sequence length of Decoder (Ty)

TRAINING ENCODER-DECODER

Training Encoder-Decoder



Training Encoder-Decoder



Next episode in EEP 596…

Attention and Transformer