CS 581

Advanced Artificial Intelligence

April 1, 2024

Announcements / Reminders

- **Please follow the Week 11 To Do List instructions (if you haven't already)**

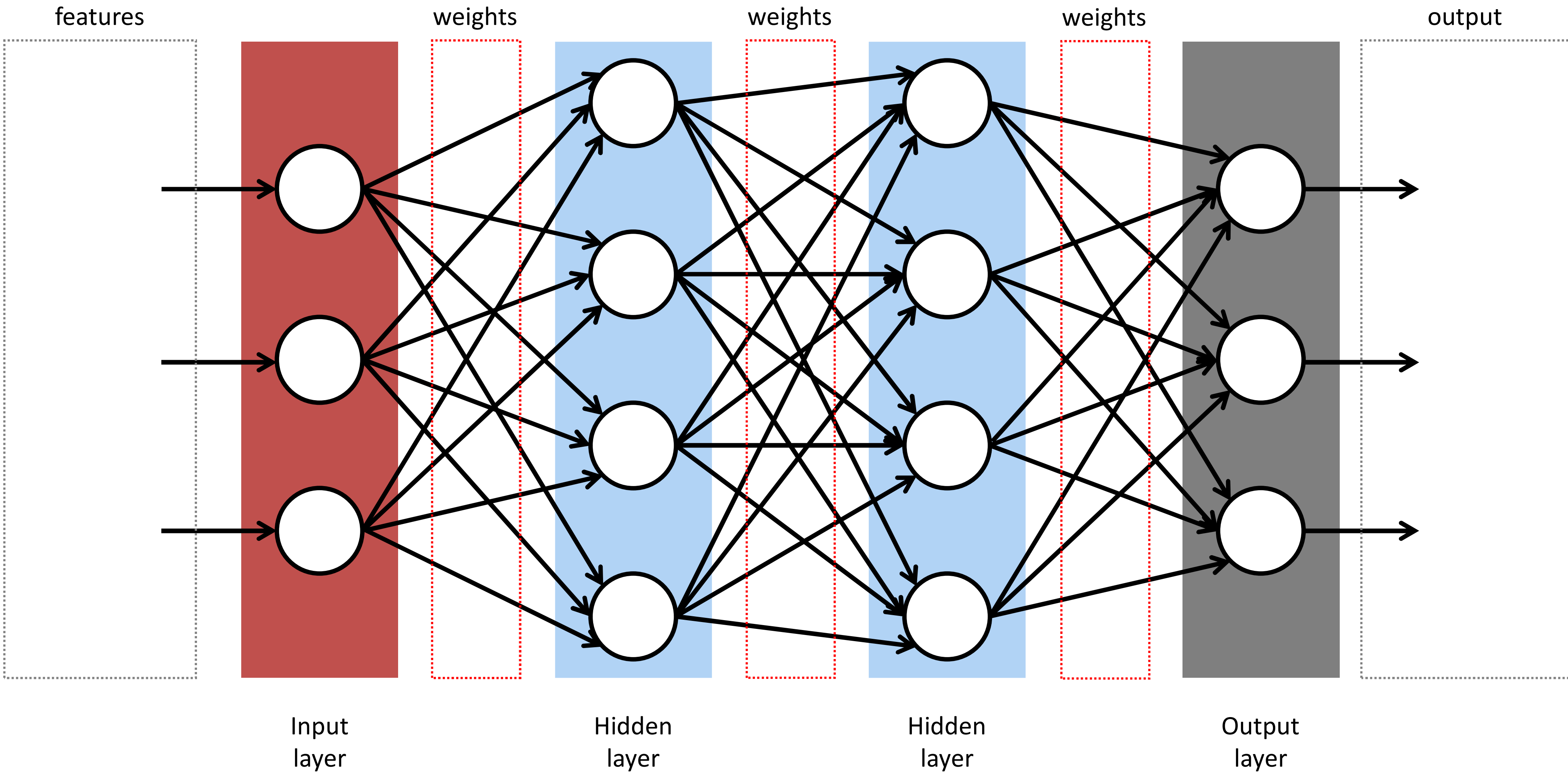
- **Programming Assignment #02 due on Sunday (04/07) at 11:59 PM CST**

Plan for Today

- Recurrent Neural Networks
  - Basic RNNs
  - Long Term Short Term Memory (LSTM)
- Seq2seq Networks

Recurrent Neural Networks

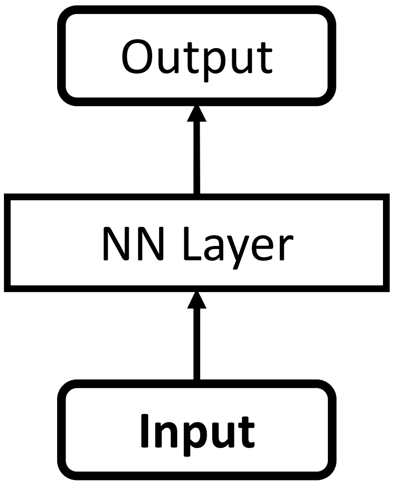
Feedforward Neural Network



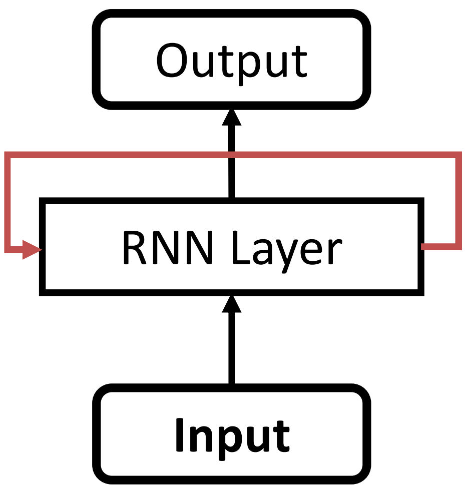
Also called (historically): multi-layer perceptron

Regular vs. Recurrent NNs

Regular NN

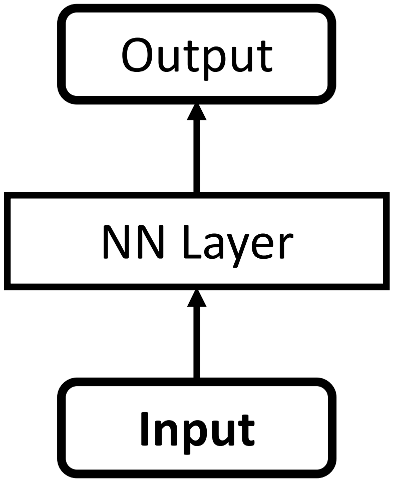


Recurrent NN



Regular vs. Recurrent NNs

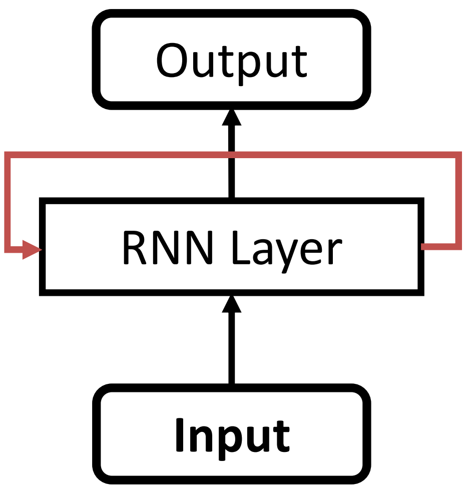
Regular NN



Does NOT have memory (does not "remember" previous state/input)

NOT suitable for sequential data

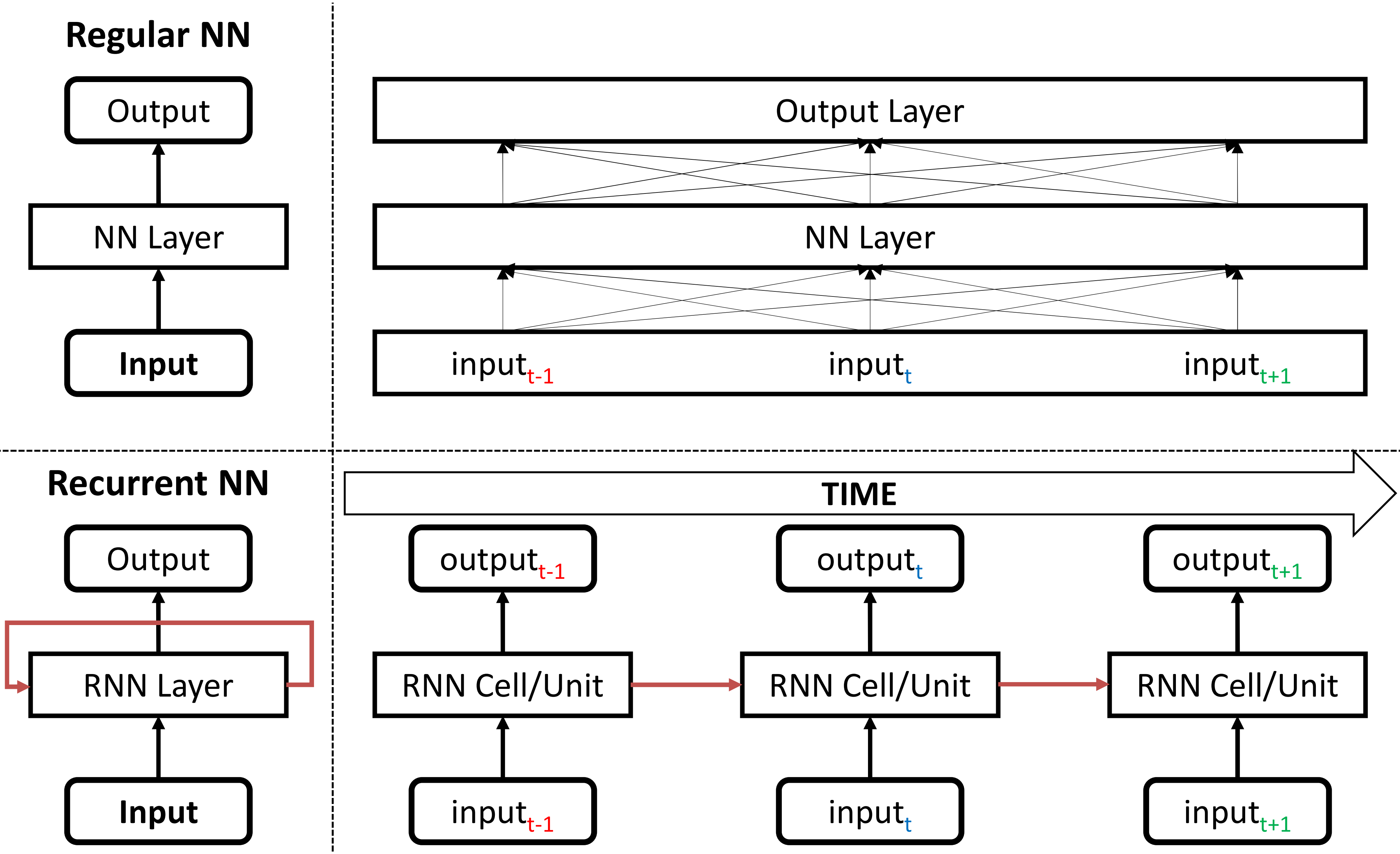
Recurrent NN



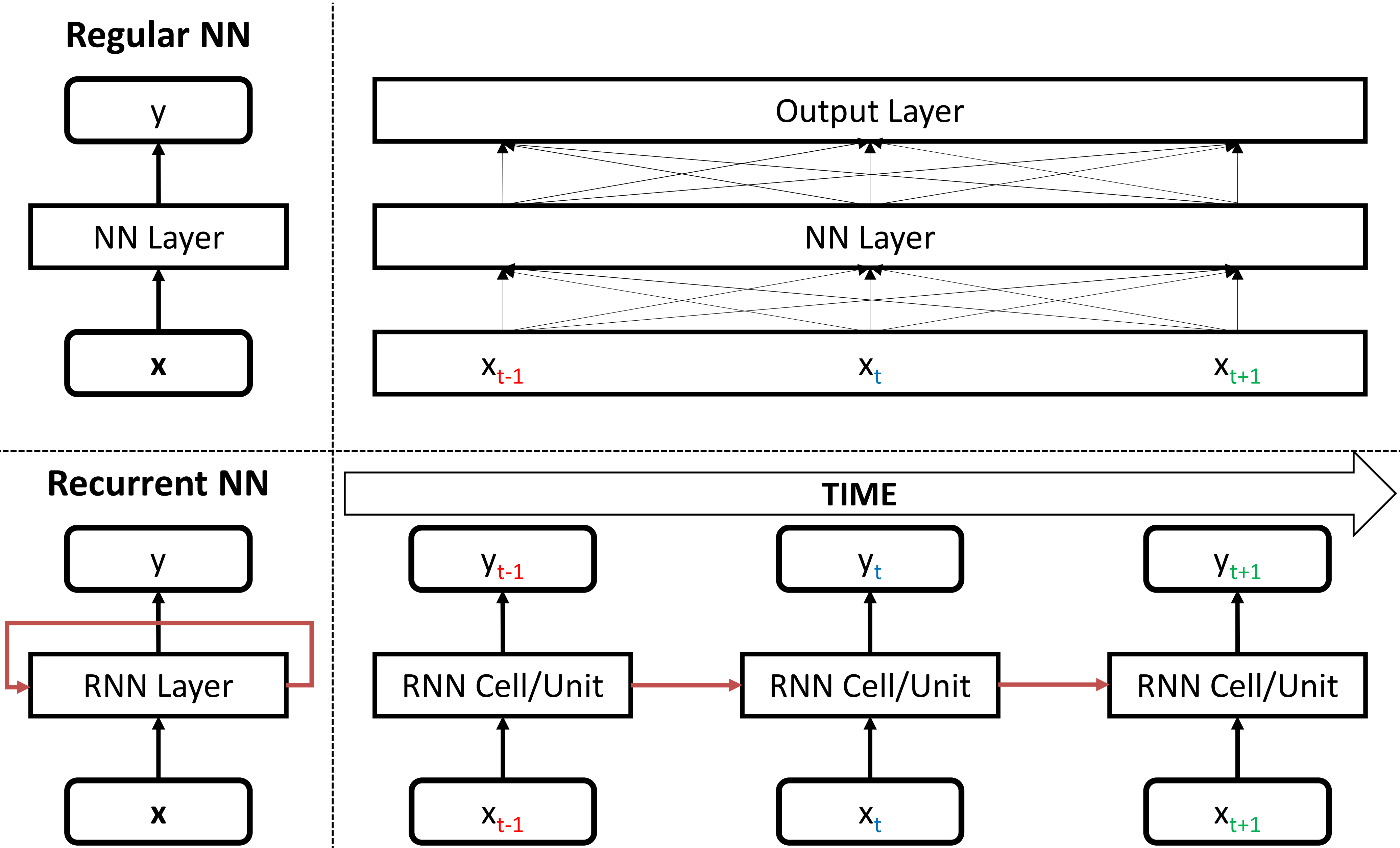
Does HAVE memory ("remembers" previous state/input)

Suitable for sequential data

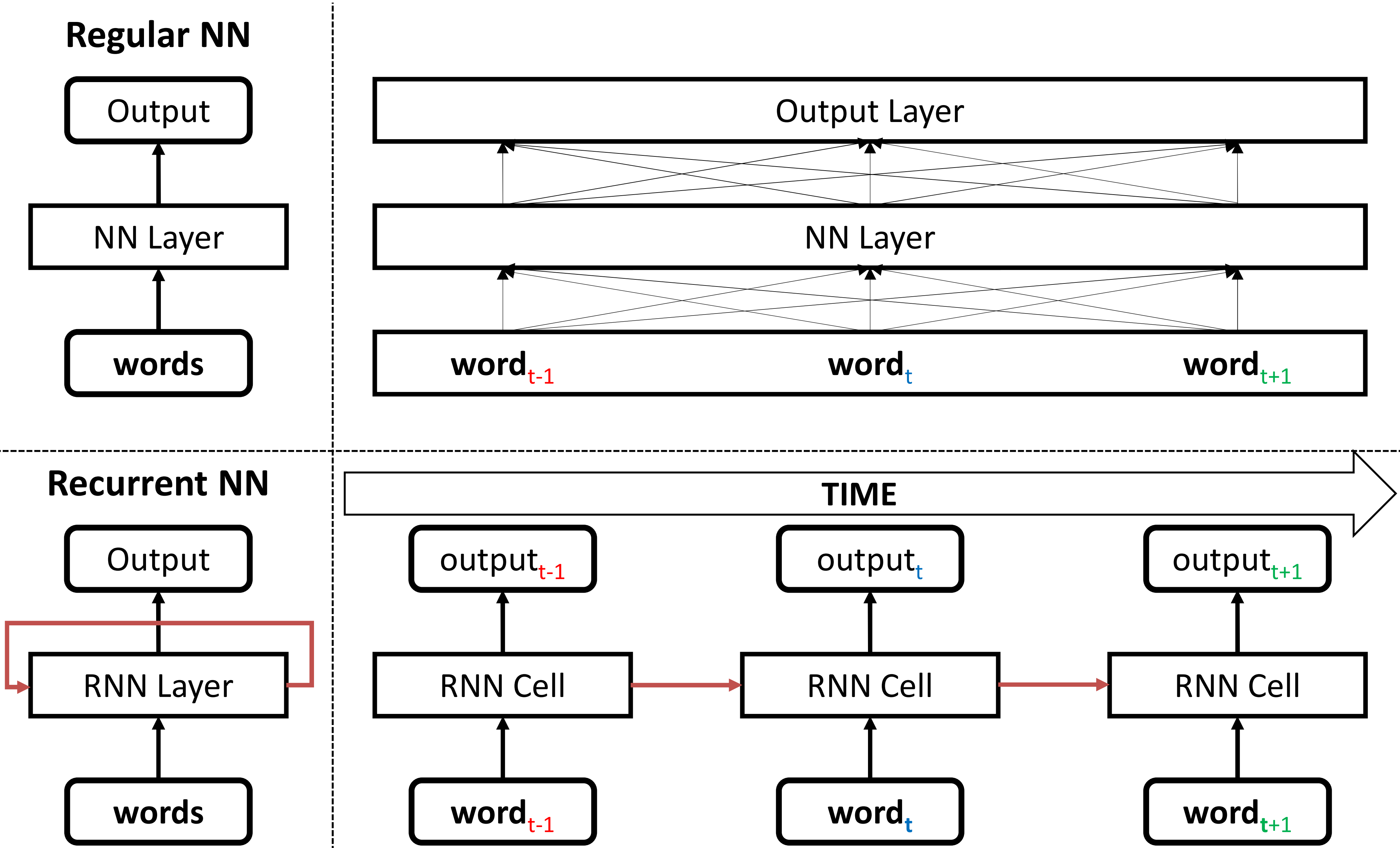
Regular vs. Recurrent NNs



Regular vs. Recurrent NNs

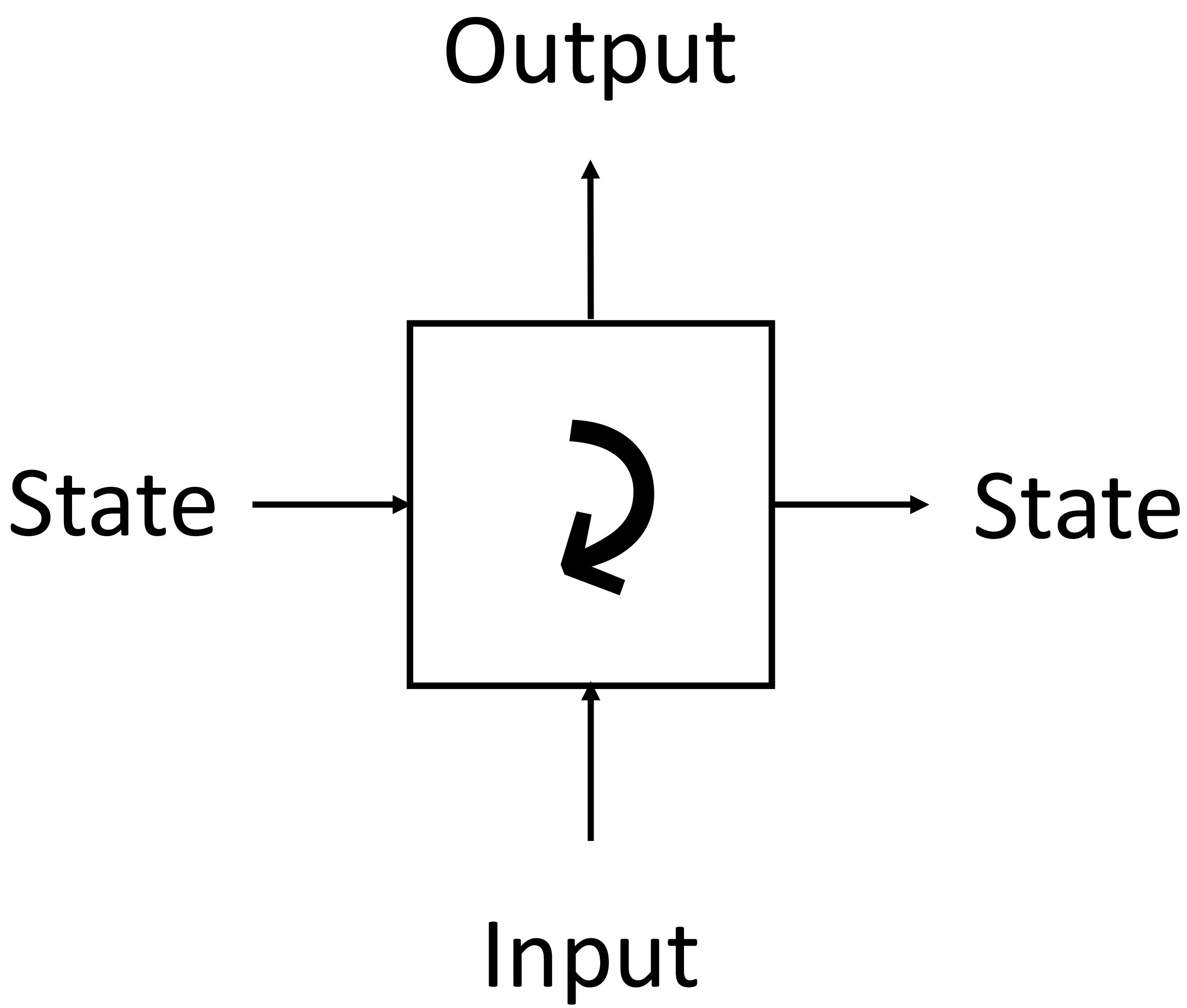


Regular vs. Recurrent NNs

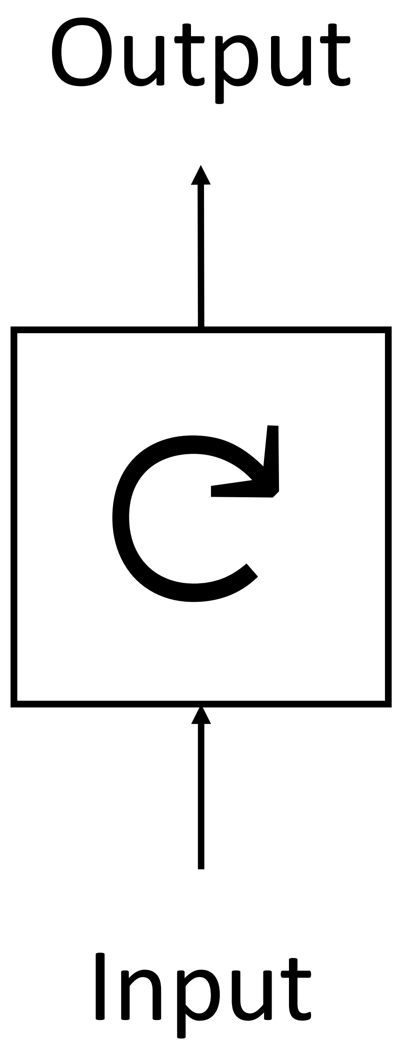


Recurrent Cells/Layers: Symbols

Recurrent Cell/Unit

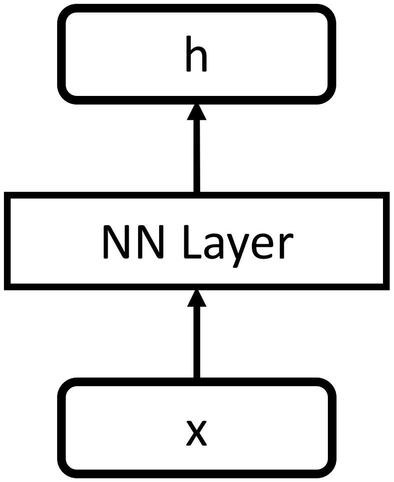


Recurrent Layer



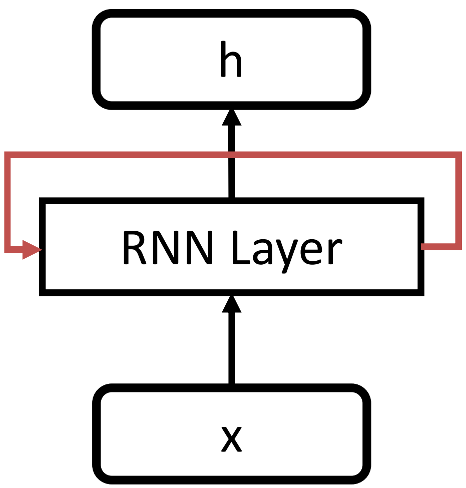
Regular vs. Recurrent NNs

Regular NN



Easier to parallelize

Recurrent NN



Difficult to parallelize

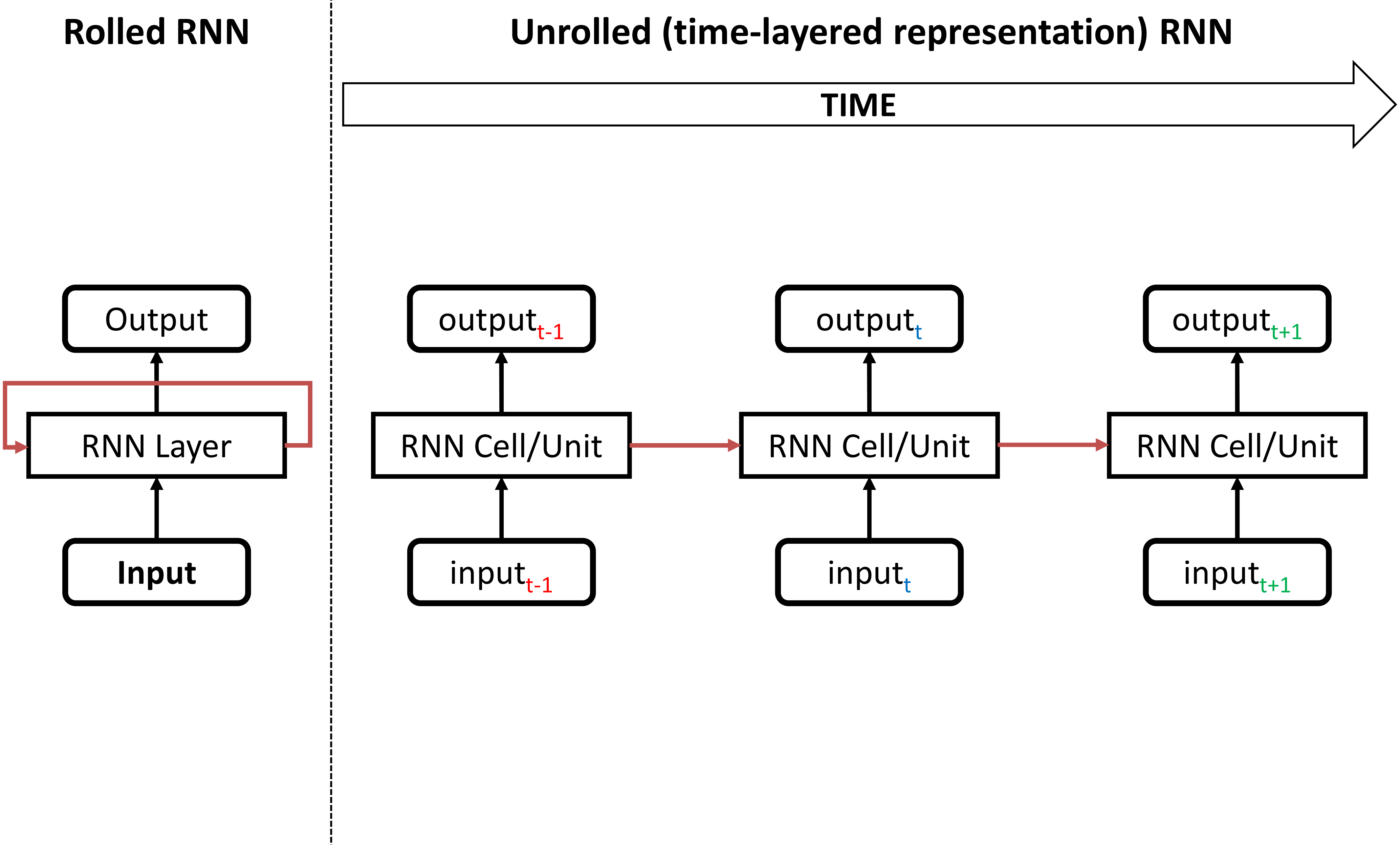
Recurrent Neural Networks (RNNs)

- Recurrent neural networks apply the same operation to the input at each time step, producing an output, but also updating an internal memory state that encodes relevant history to be used in prediction

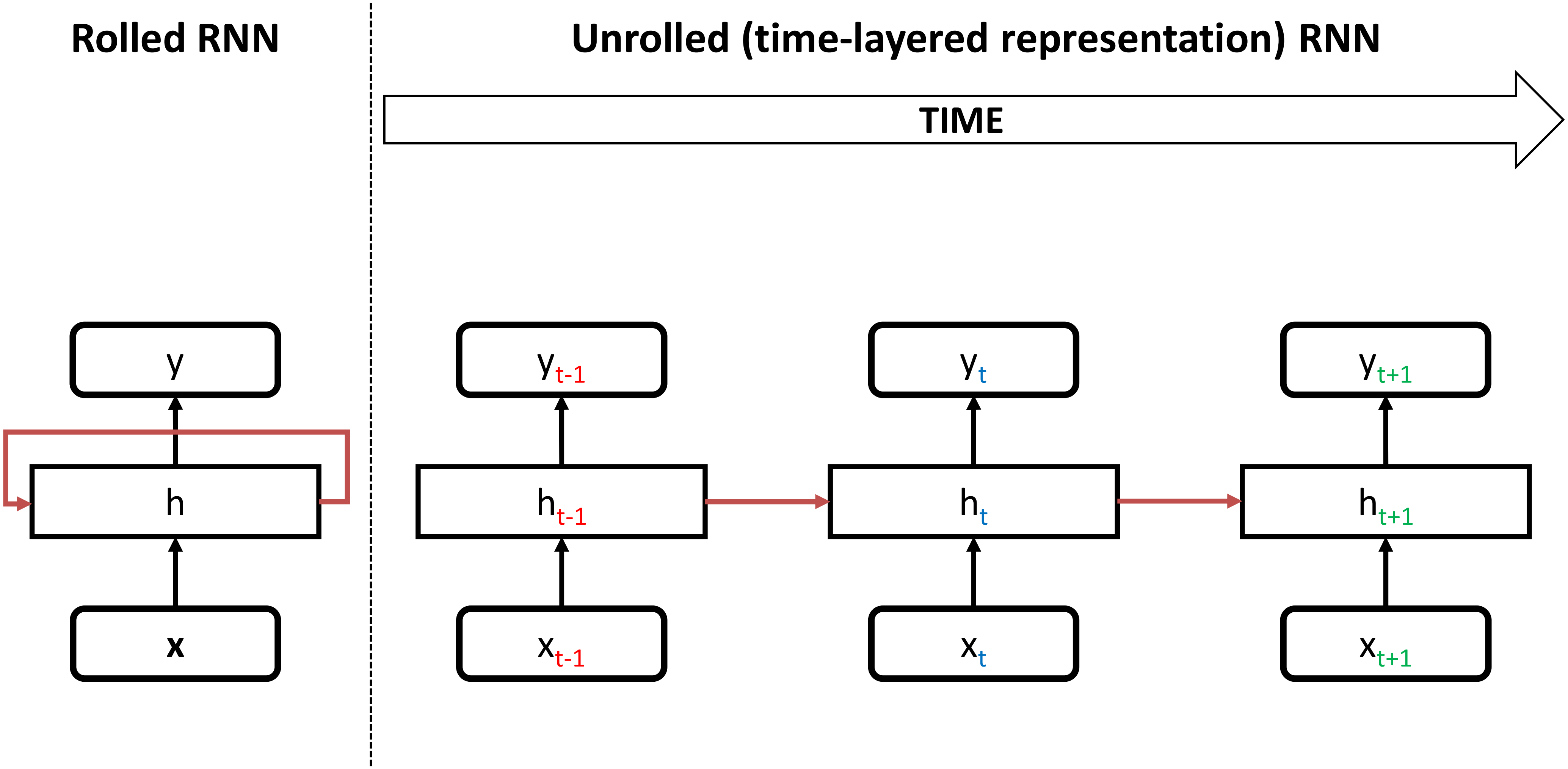
- This memory state can allow distant information to influence the prediction made for a given word/label

- Because the memory state is transferred from time step to time step, the network is intrinsically sequential – it cannot be effectively parallelized

Rolled vs. Unrolled RNN

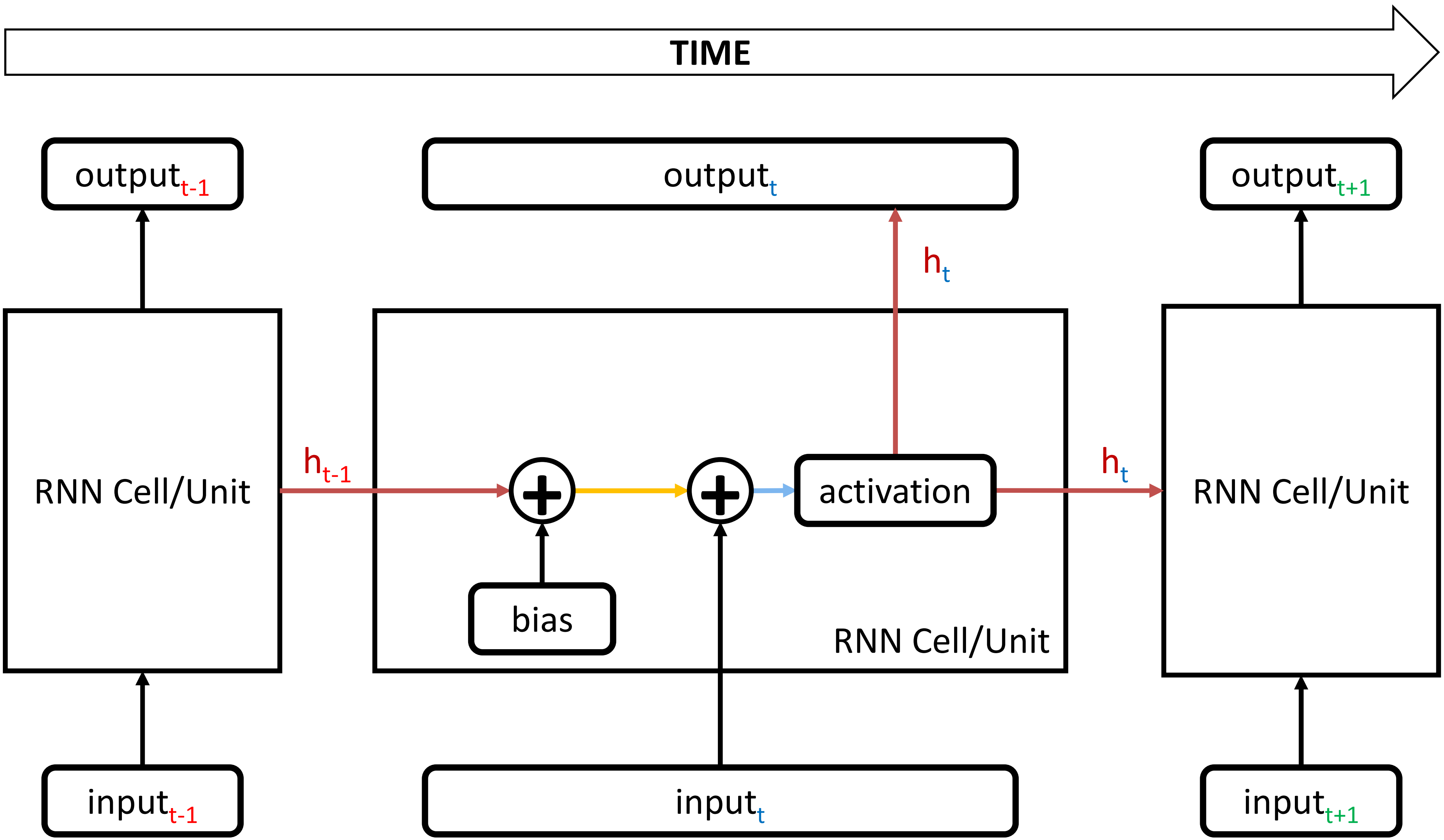


Rolled vs. Unrolled RNN

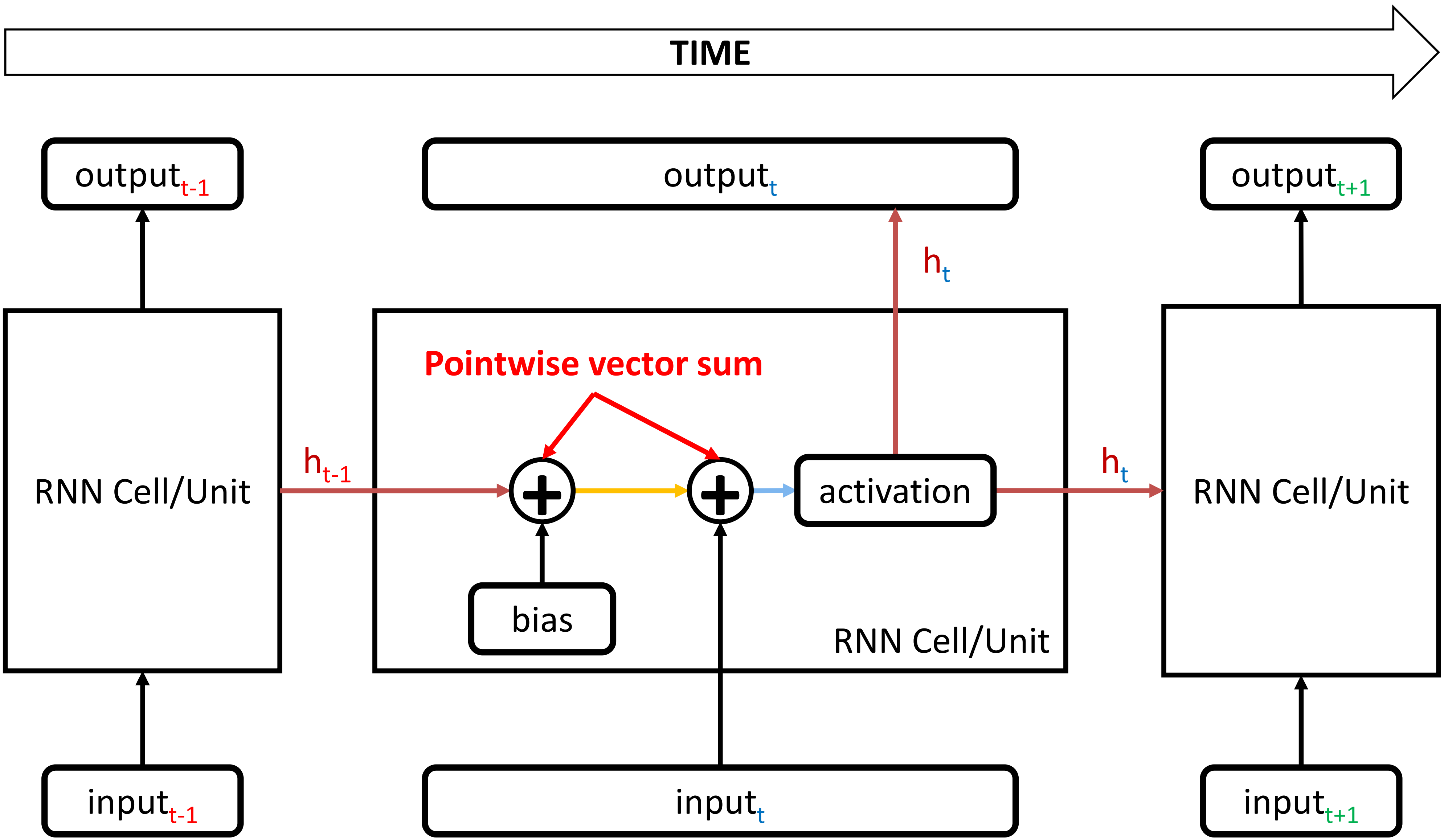


x_i – single input/feature (can be a scalar or a **vector**)
h_i – single memory/hidden representation/state (can be a scalar or a **vector**)
y_i – single output (can be a scalar or a **vector**)

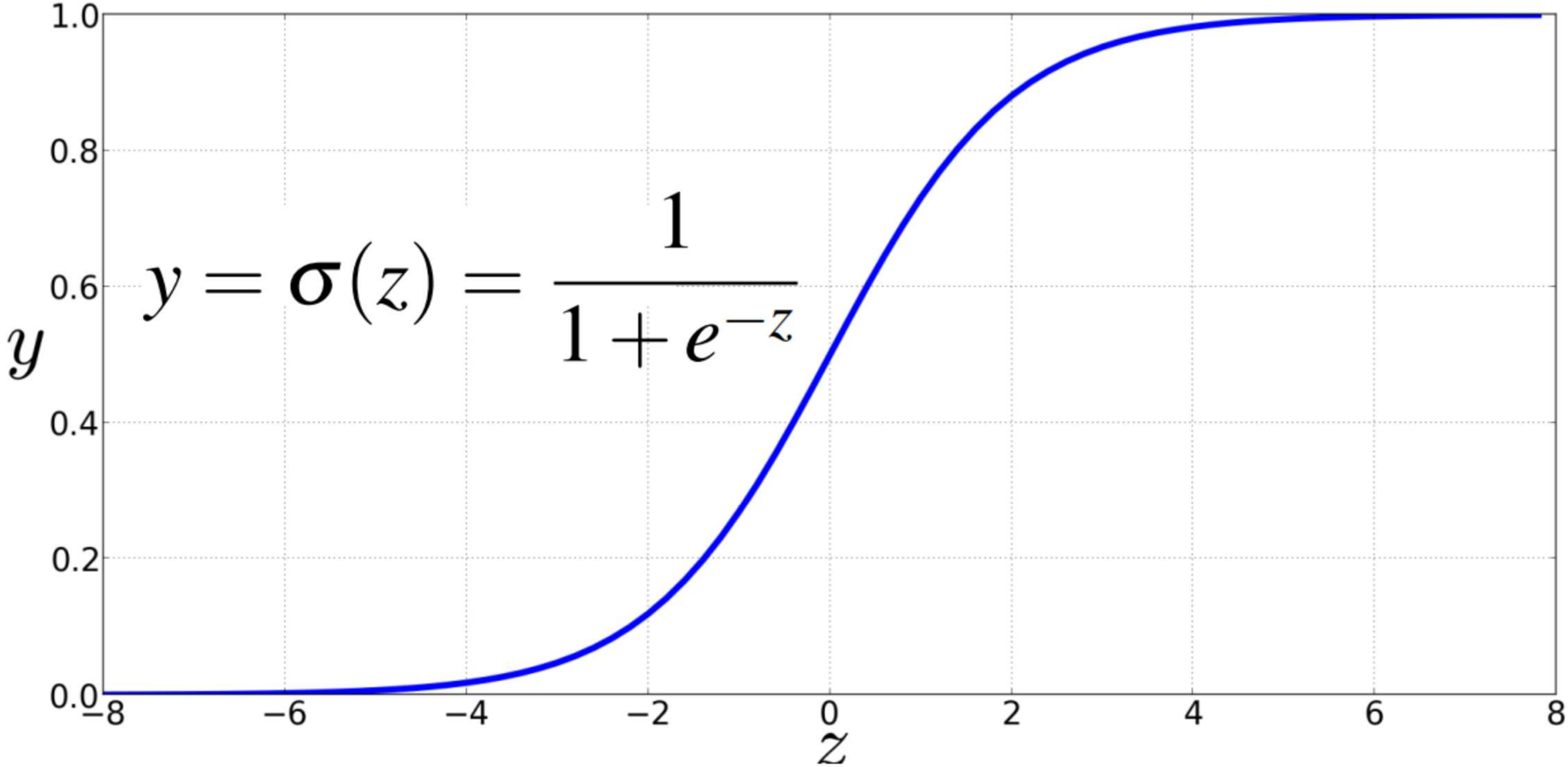
RNN Cell/Unit



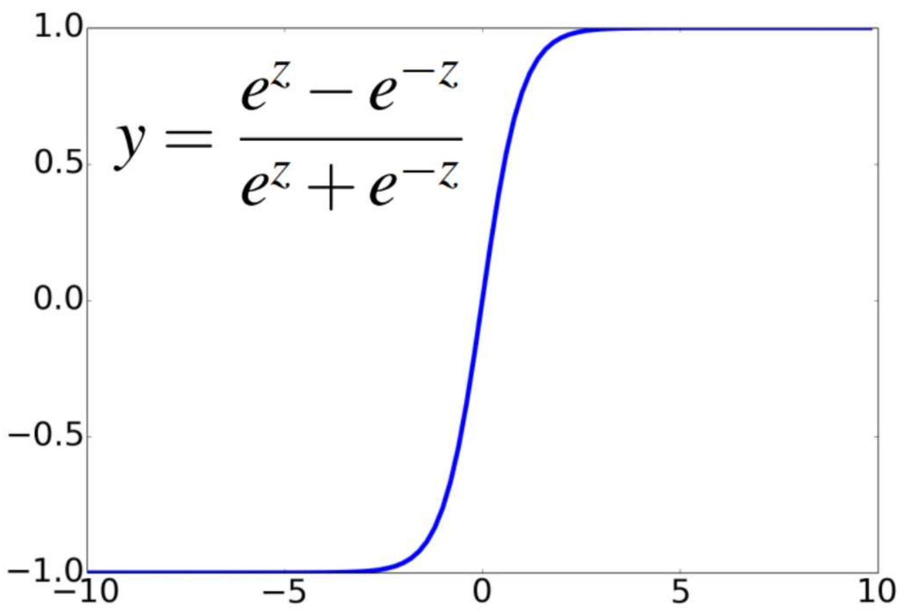
RNN Cell/Unit



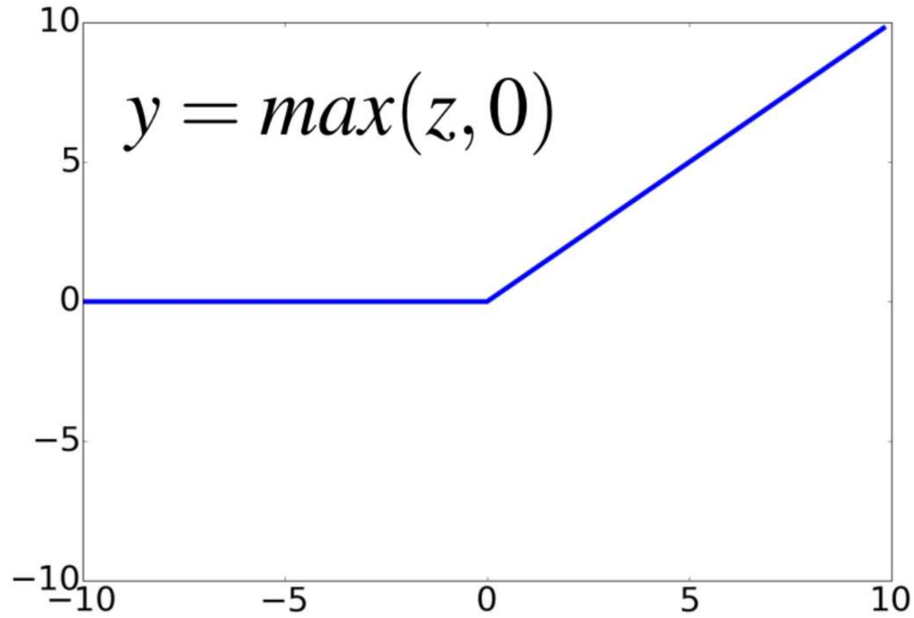
Sigmoid / Logistic Activation Function



Other Nonlinear Activation Functions

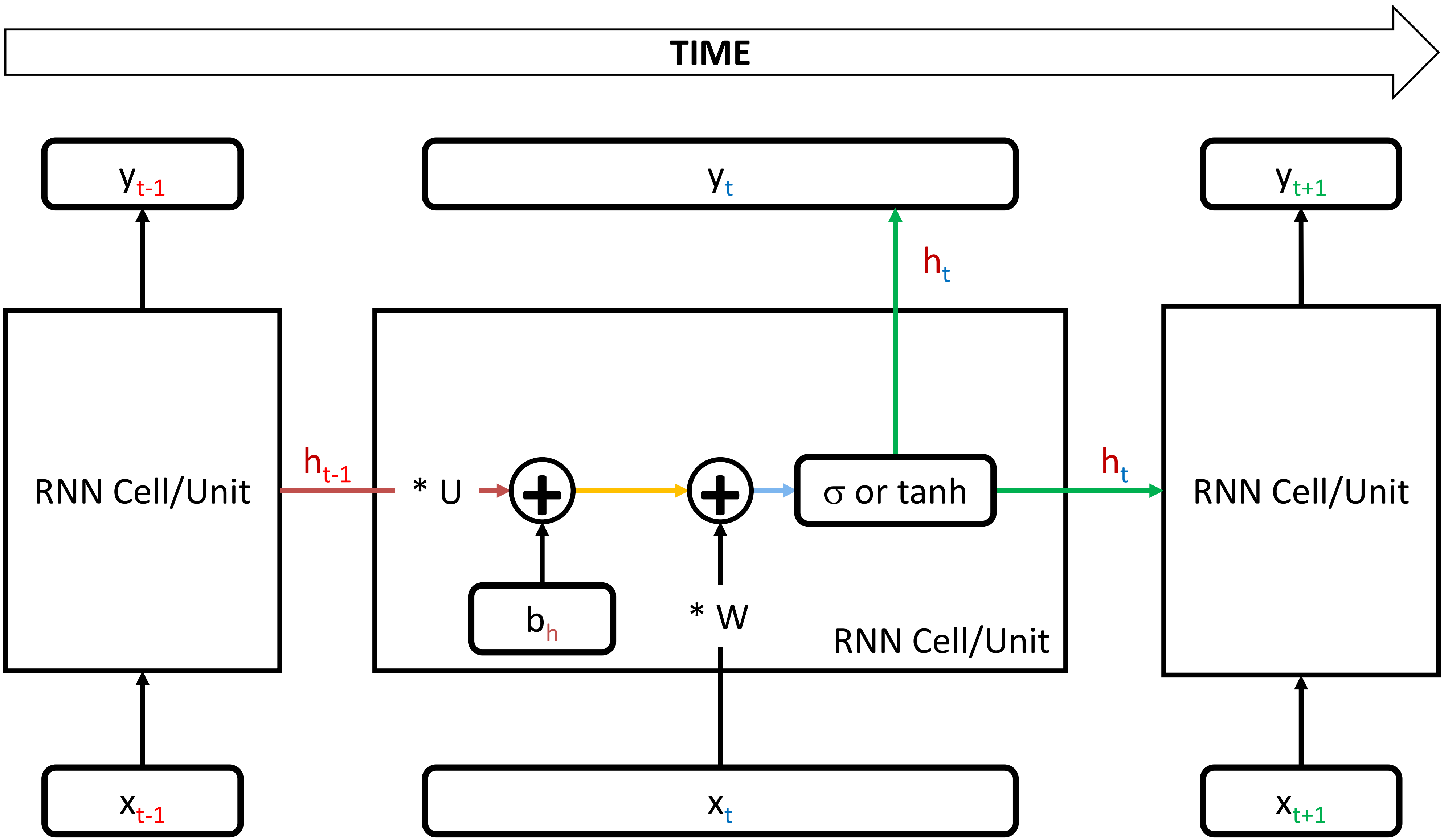


tanh
(hyperbolic tangent)

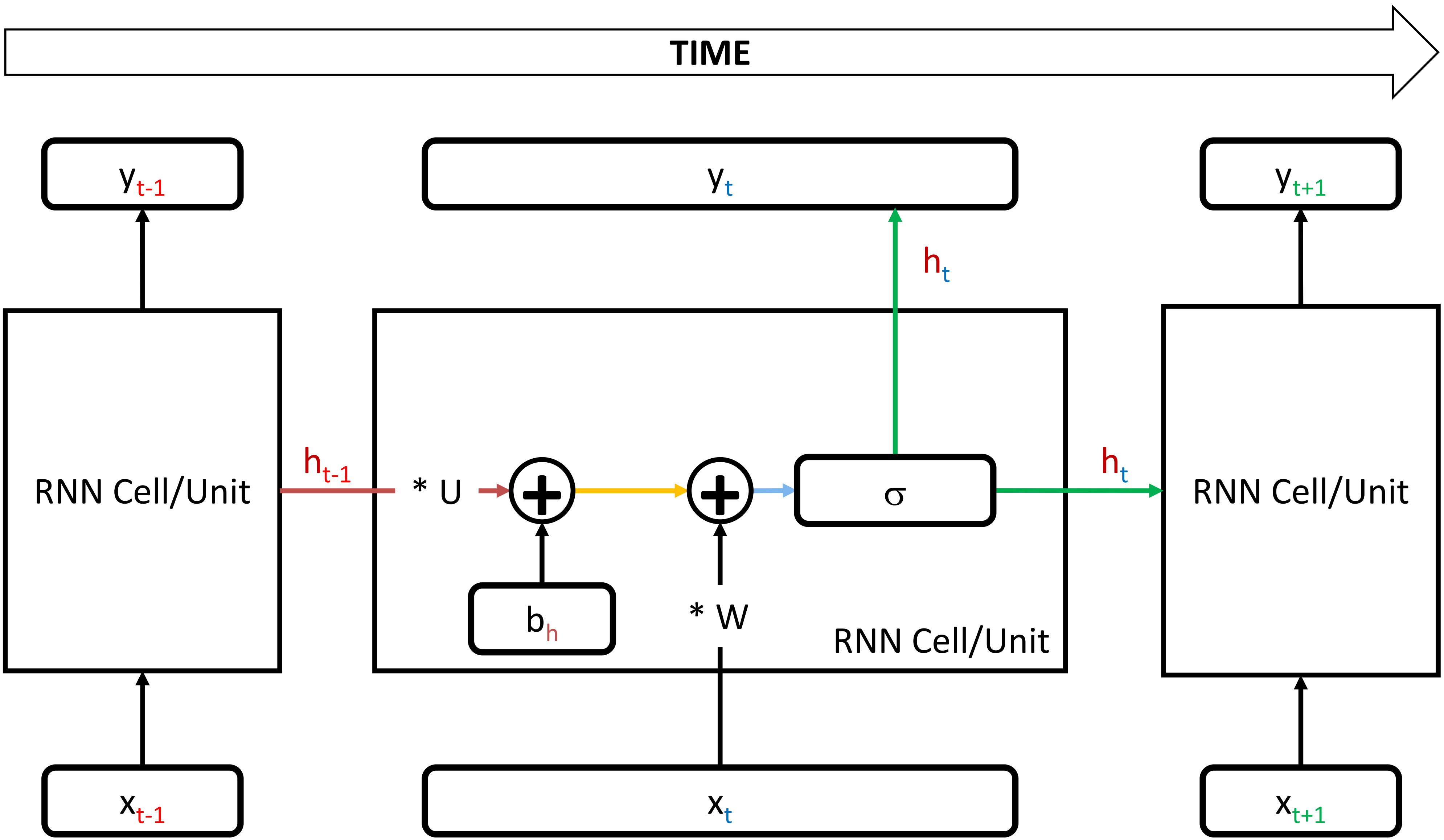


ReLU
(Rectified Linear Unit)

RNN Cell/Unit

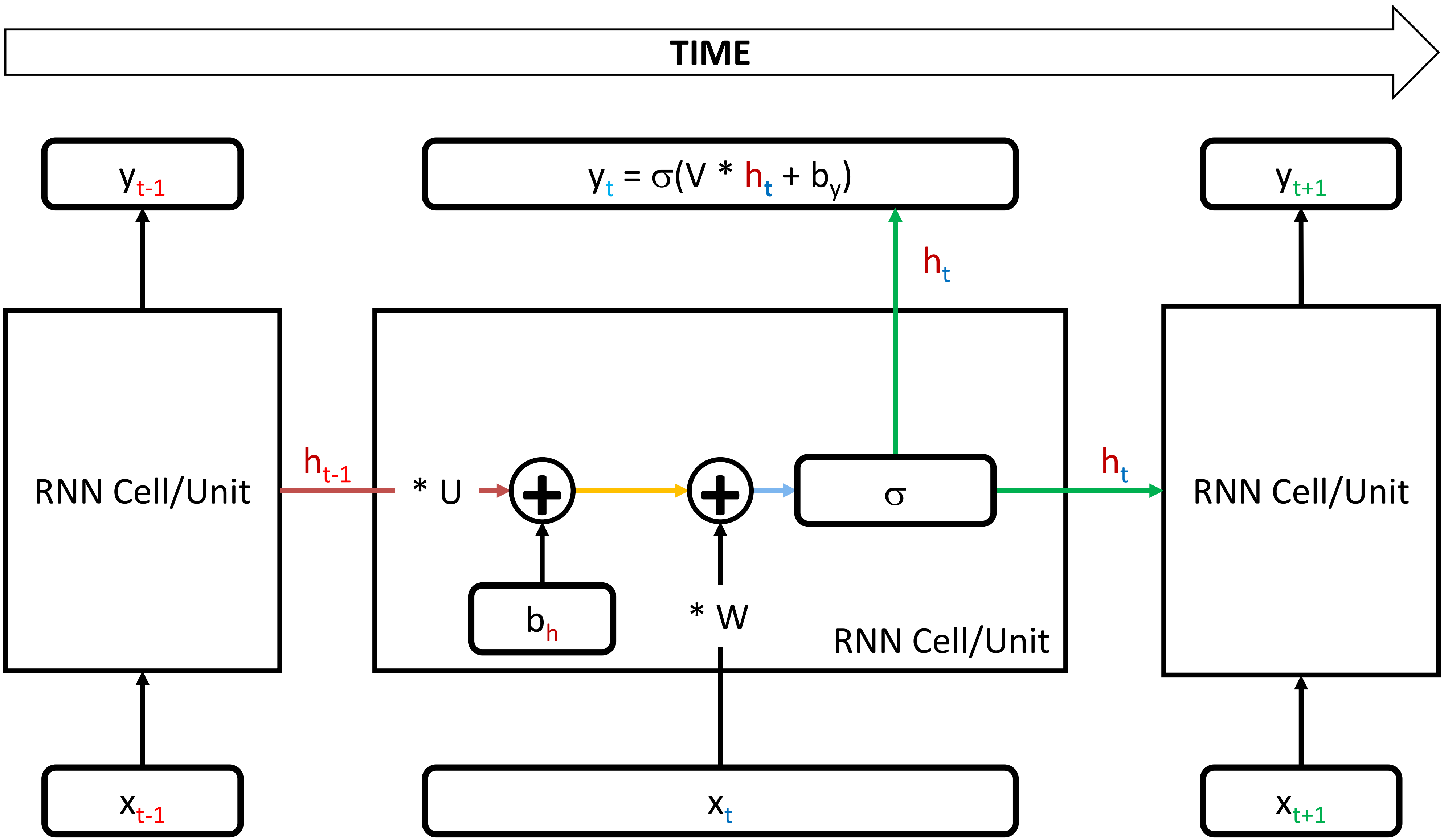


RNN Cell/Unit



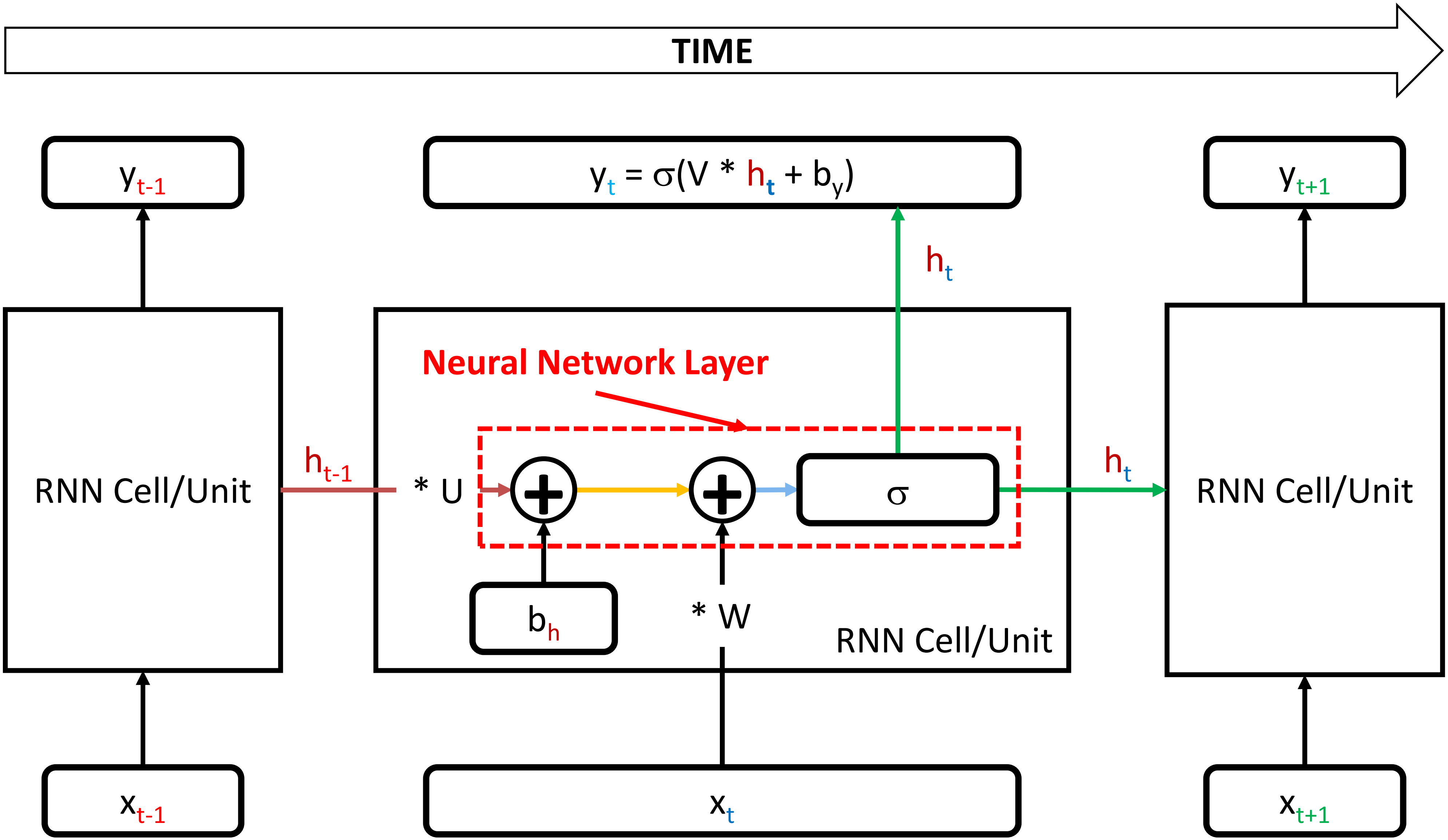
$$h_t = \sigma(W * x_t + U * h_{t-1} + b_h)$$

RNN Cell/Unit



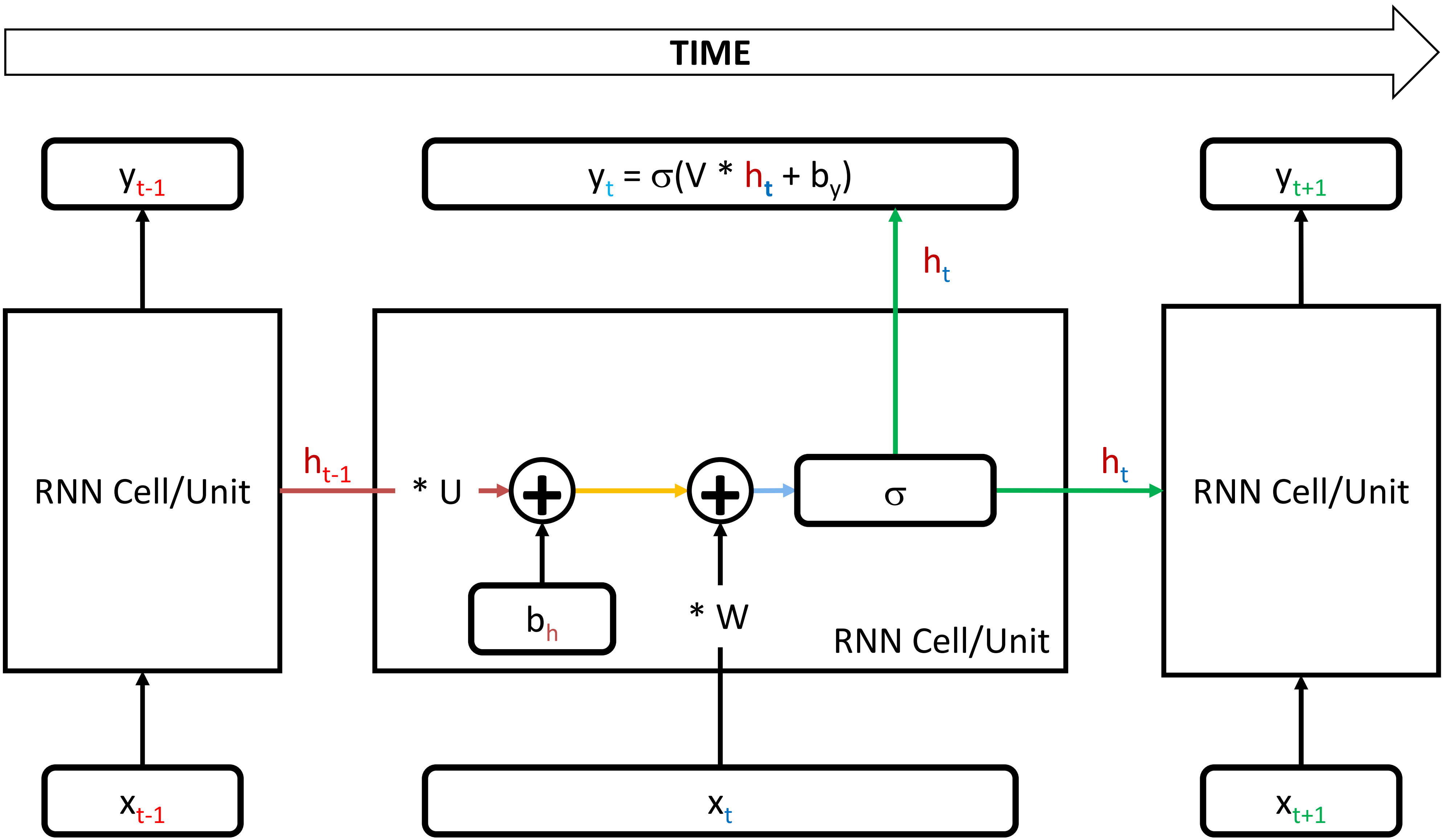
$$h_t = \sigma(W * x_t + U * h_{t-1} + b_h)$$

RNN Cell/Unit



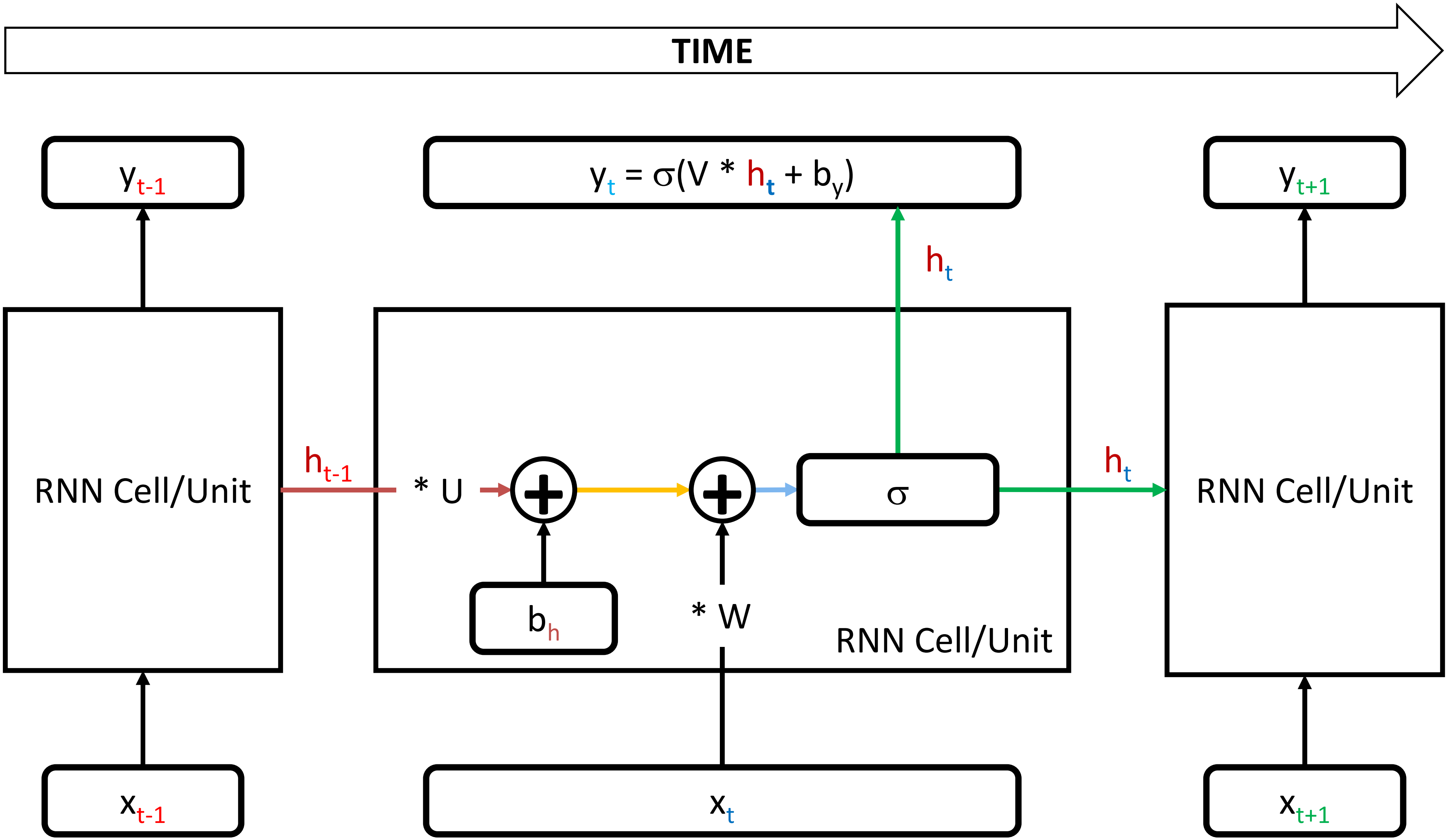
$$h_t = \sigma(W * x_t + U * h_{t-1} + b_h)$$

RNN Cell/Unit



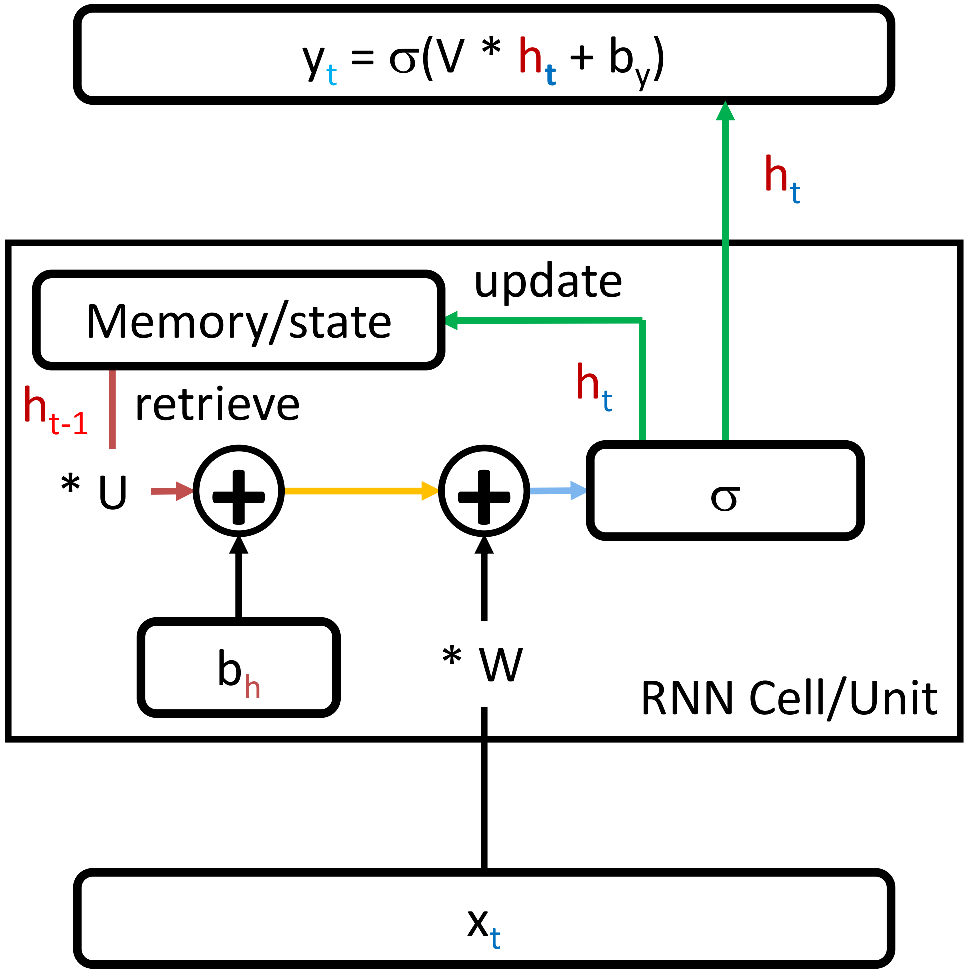
W: Input weight matrix | U: Recurrent weight matrix | V: Output weight matrix | b_y: output bias

RNN Cell/Unit



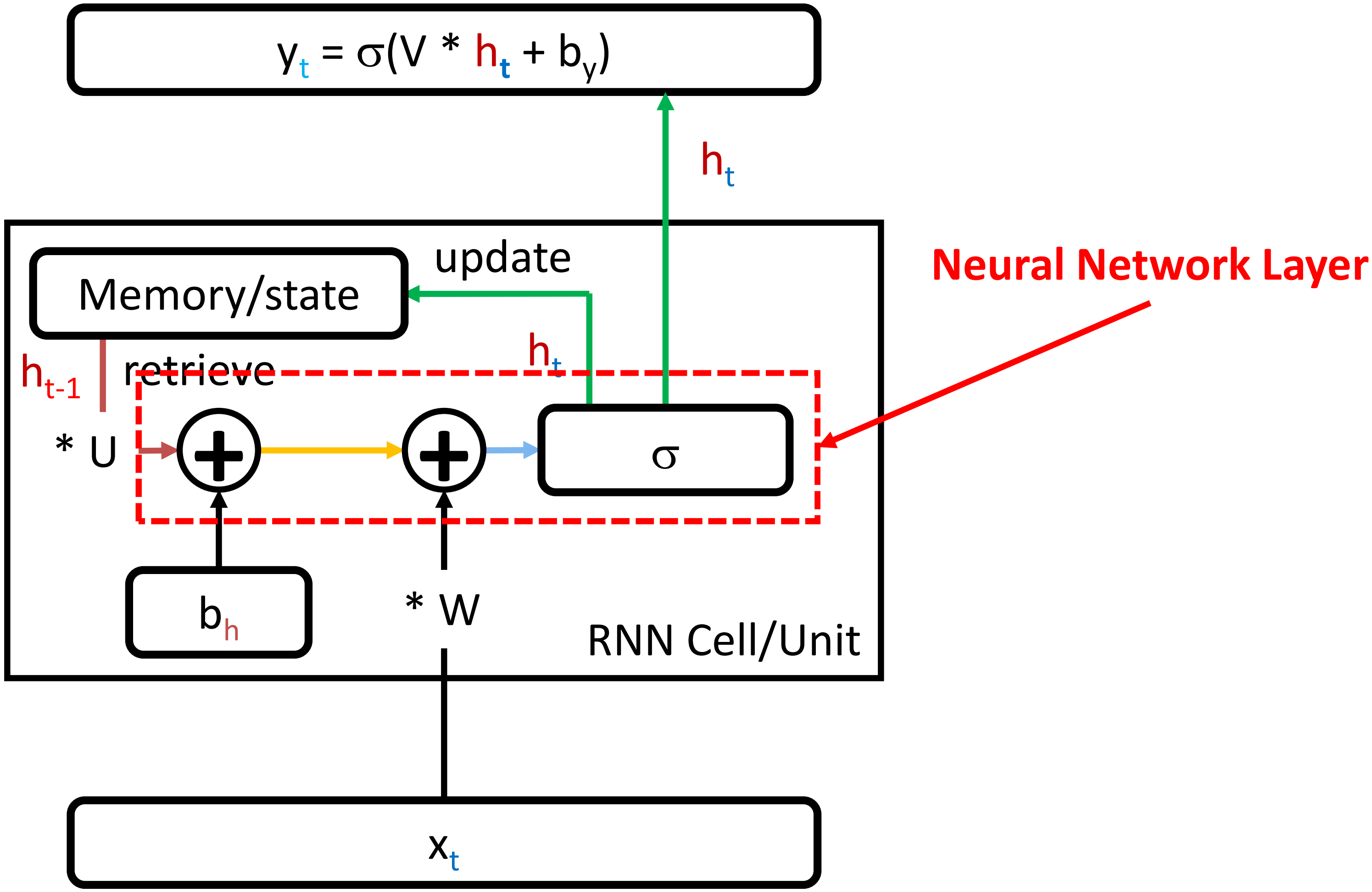
x_i: size d | h_i: size p | y_i: size d | W: p x d | U: p x p | V: d x p | b_h: size p | b_y: size d

In Practice: RNN Cell/Unit



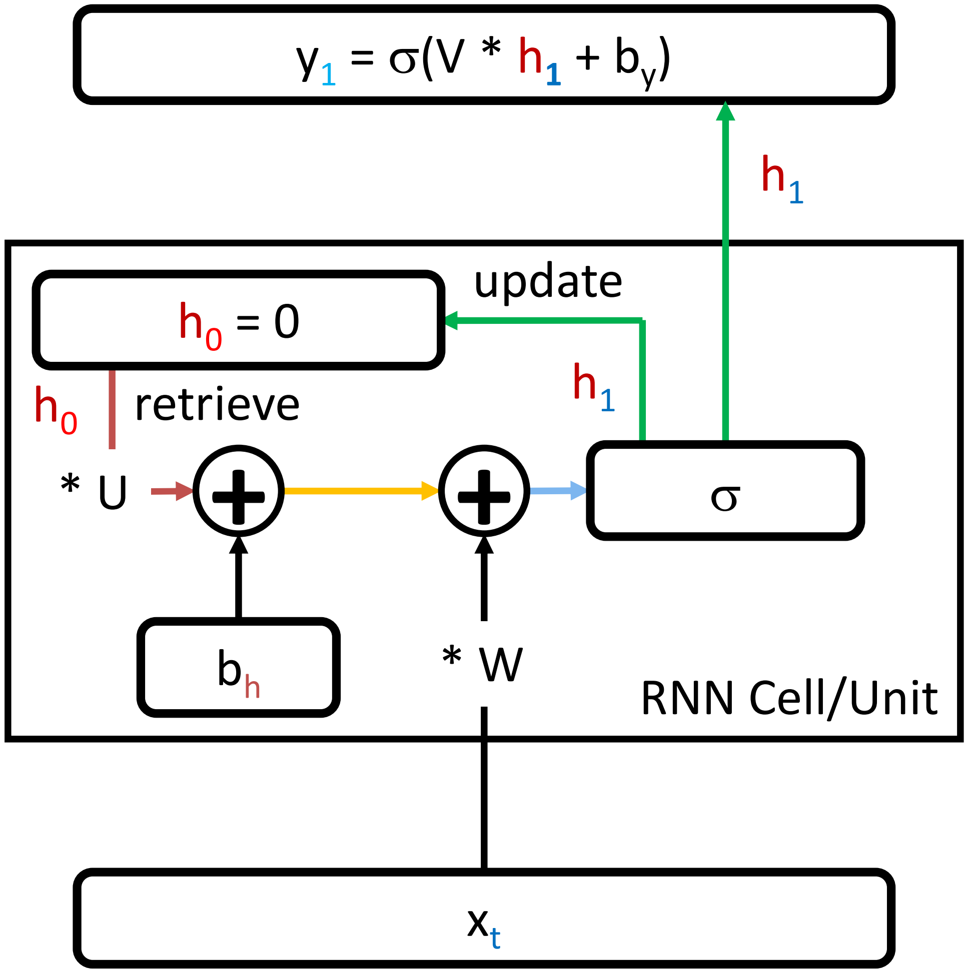
$h_t = \sigma(W * x_t + U * h_{t-1} + b_h)$

In Practice: RNN Cell/Unit



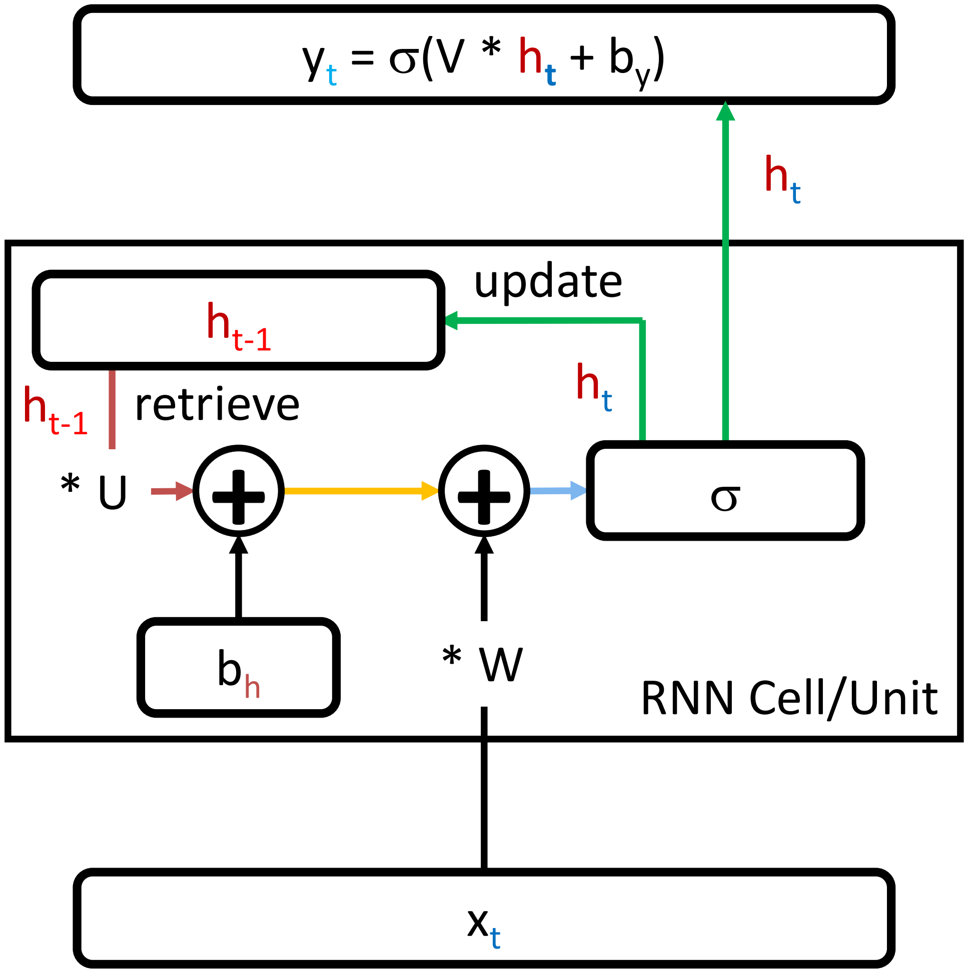
$$h_t = \sigma(W * x_t + U * h_{t-1} + b_h)$$

In Practice: RNN Cell/Unit | T = 1



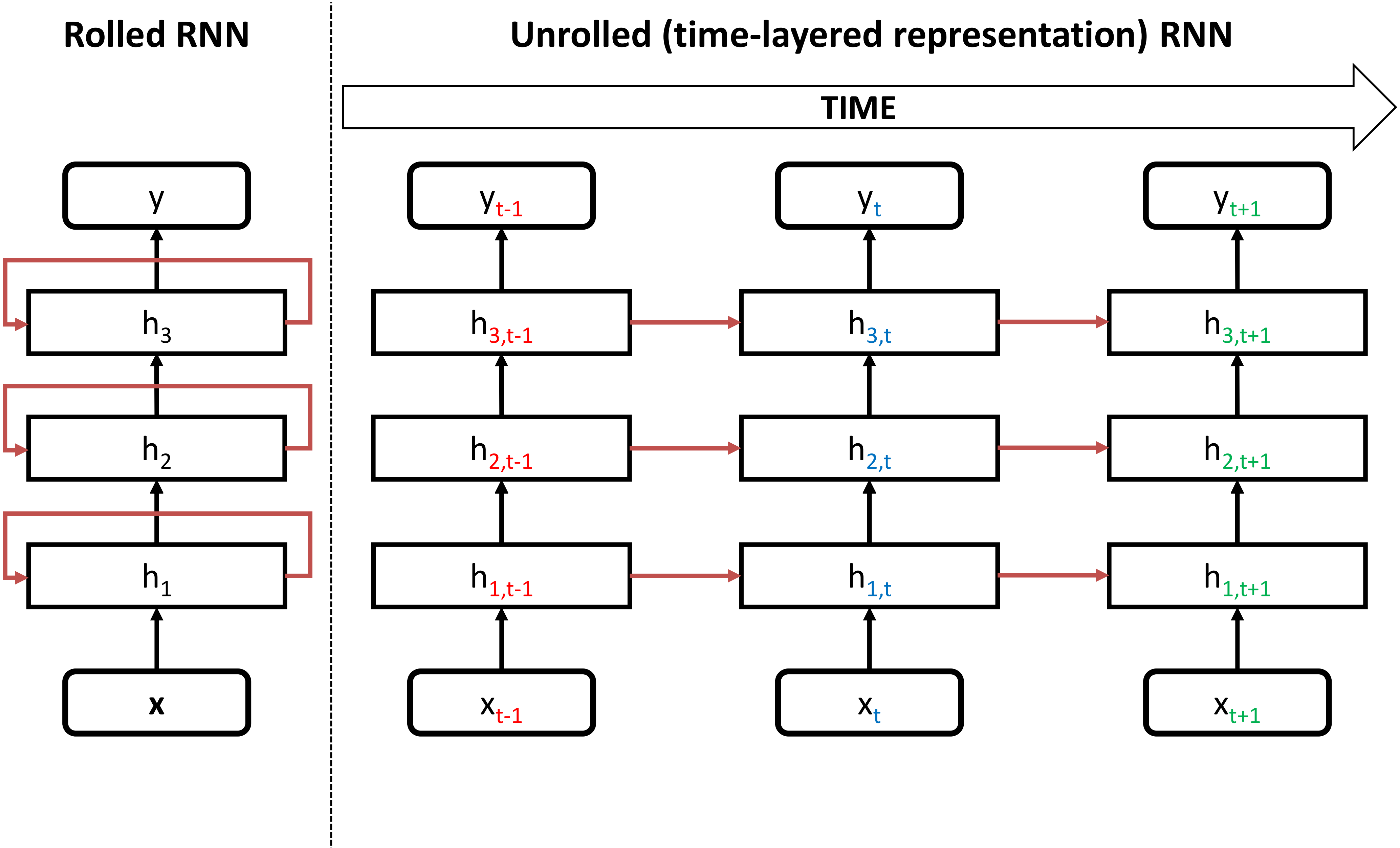
$h_1 = \sigma(W * x_1 + U * h_0 + b_h) = h_1 = \sigma(W * x_1 + U * \mathbf{0} + b_h) = h_1 = \sigma(W * x_1 + b_h)$

In Practice: RNN Cell/Unit | T = t

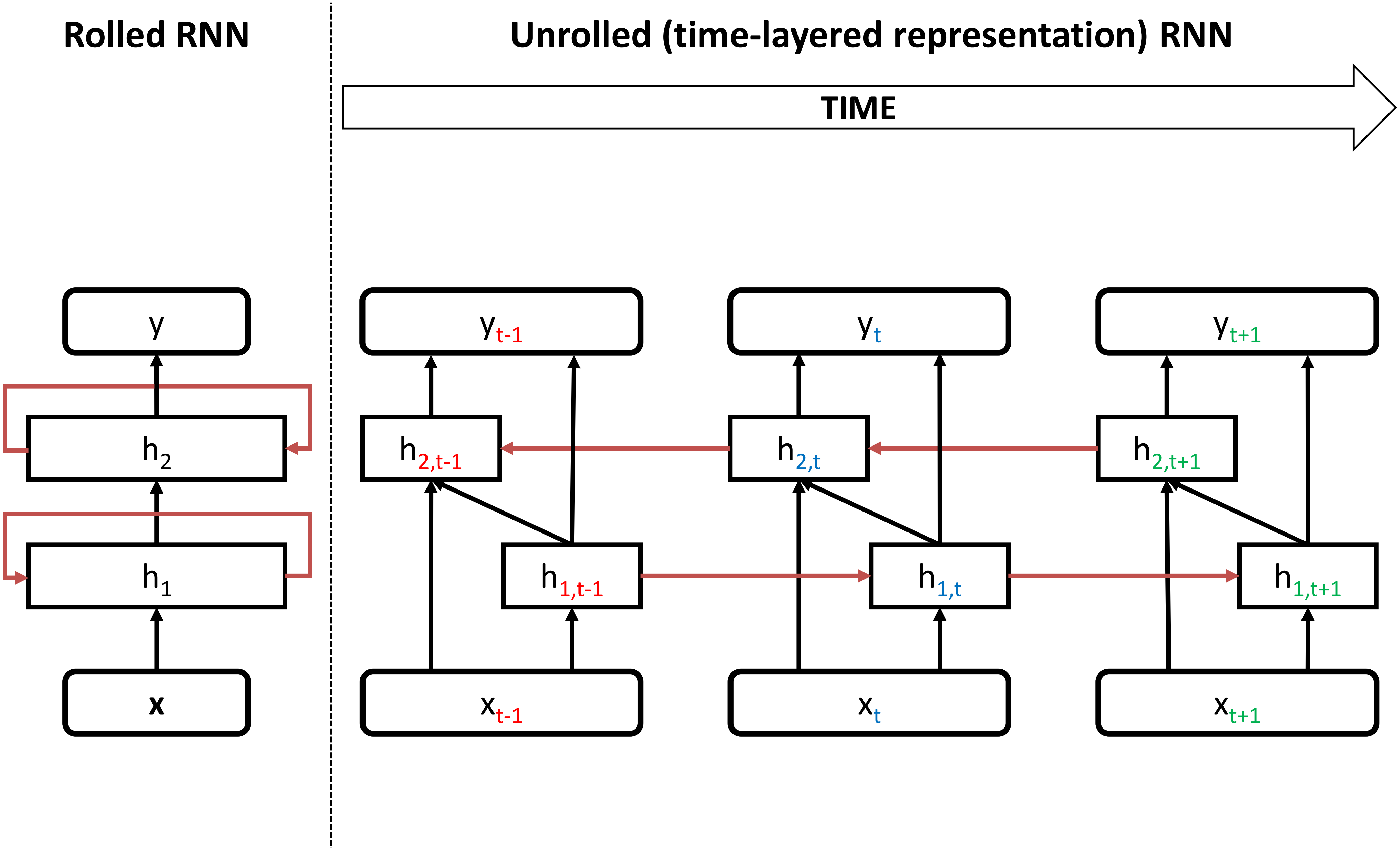


$h_t = \sigma(W * x_t + U * h_{t-1} + b_h)$

In Practice: Multi-Layer RNNs

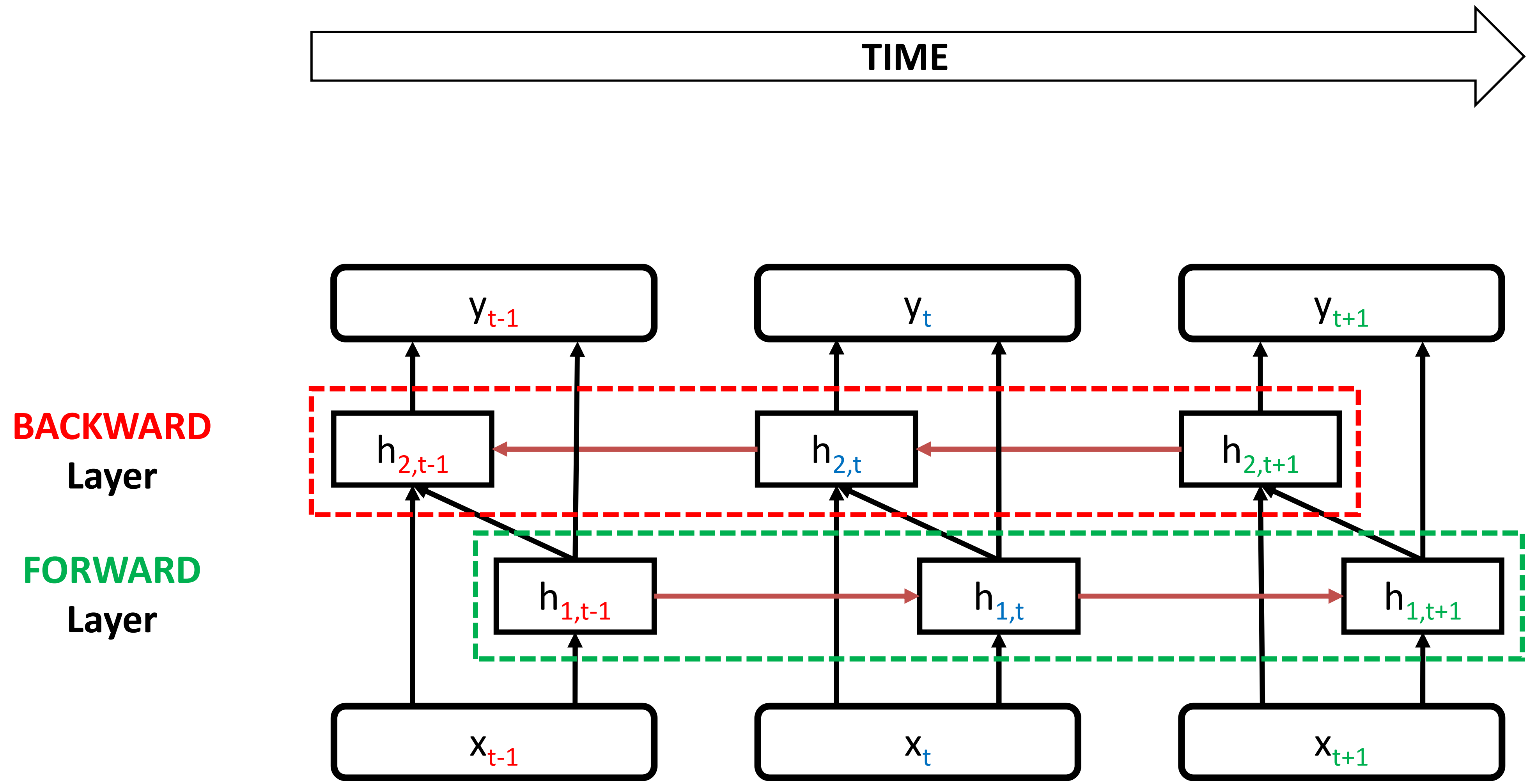


In Practice: Bi-directional RNNs



In Practice: Bi-directional RNNs

Unrolled (time-layered representation) RNN



RNN: Input - Output

$$y_5 = \sigma(Vh_5 + b_y)$$

$$\sigma(Wx_5 + Uh_4 + b_h)$$

$$\sigma(Wx_4 + Uh_3 + b_h)$$

$$\sigma(Wx_3 + Uh_2 + b_h)$$

$$\sigma(Wx_2 + Uh_1 + b_h)$$

$$\sigma(Wx_1 + U\mathbf{0} + b_h)$$

Training Neural Networks: Intuition

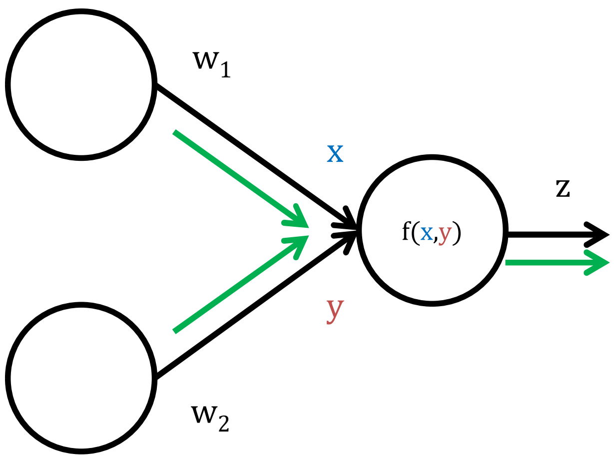
For every training tuple (x ,y) = (feature vector, label)

- **Run forward computation to find estimate $\hat{y}$**
- **Run backward computation to update weights:**
  - **For every output node**
    - **Compute loss L between true y and the estimated $\hat{y}$**
    - **For every weight w from hidden layer to the output layer**
    - **Update the weight**
  - **For every hidden node**
    - **Assess how much blame it deserves for the current answer**
    - **For every weight w from input layer to the hidden layer**
      - **Update the weight**

Back-propagation

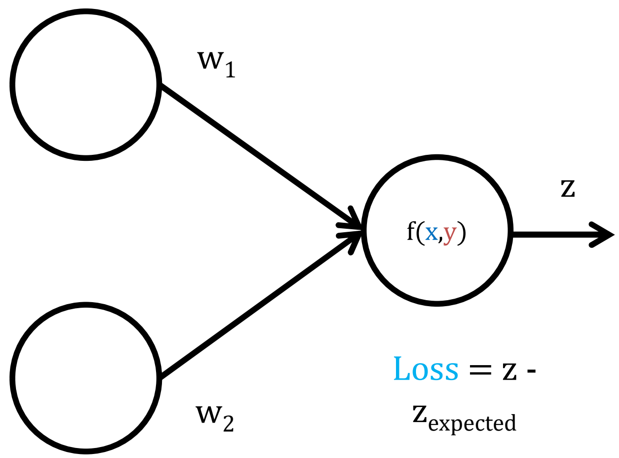
Feed forward | **Evaluate Loss** | **Back-propagation**

Feed forward



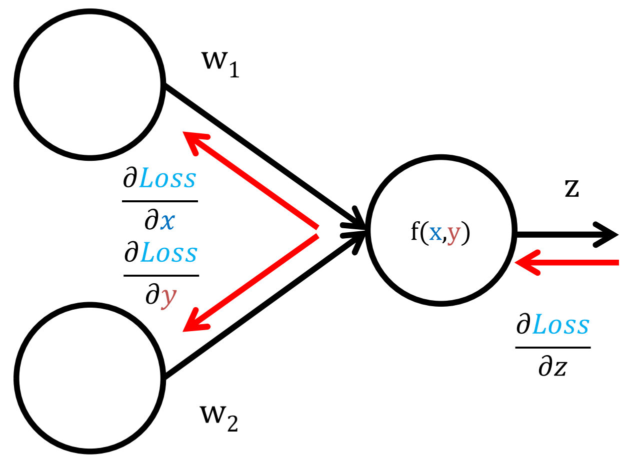
Feed a **labeled sample** through the network

Evaluate Loss



How "incorrect" is the result compare to the label?

Back-propagation



Update weights (use **Gradient Descent**)

Gradients and Learning Rate

- The value of the gradient (slope in our example) $\frac{d}{dw}L(f(x;w),y)$ weighted by a **learning rate** η

- Higher learning rate means move **w** faster

$$w^{t+1} = w^t - \eta \frac{d}{dw}L(f(x;w),y)$$

Vanishing and Exploding Gradients

- As we've seen, information needs to travel a long way in an RNN to get from the error signal / loss function (y) to some inputs (xi). By the chain rule of differentiation, the gradient of the loss function will have the form

$$W \times \sigma'(z_1) \times U \times \sigma'(z_2) \times U \times \sigma'(z_3) \cdots$$

- Vanishing gradients: Elements of U are less than one, and gradients drop off to zero
- Exploding gradients: Elements of U are greater than one and gradients increase without limit

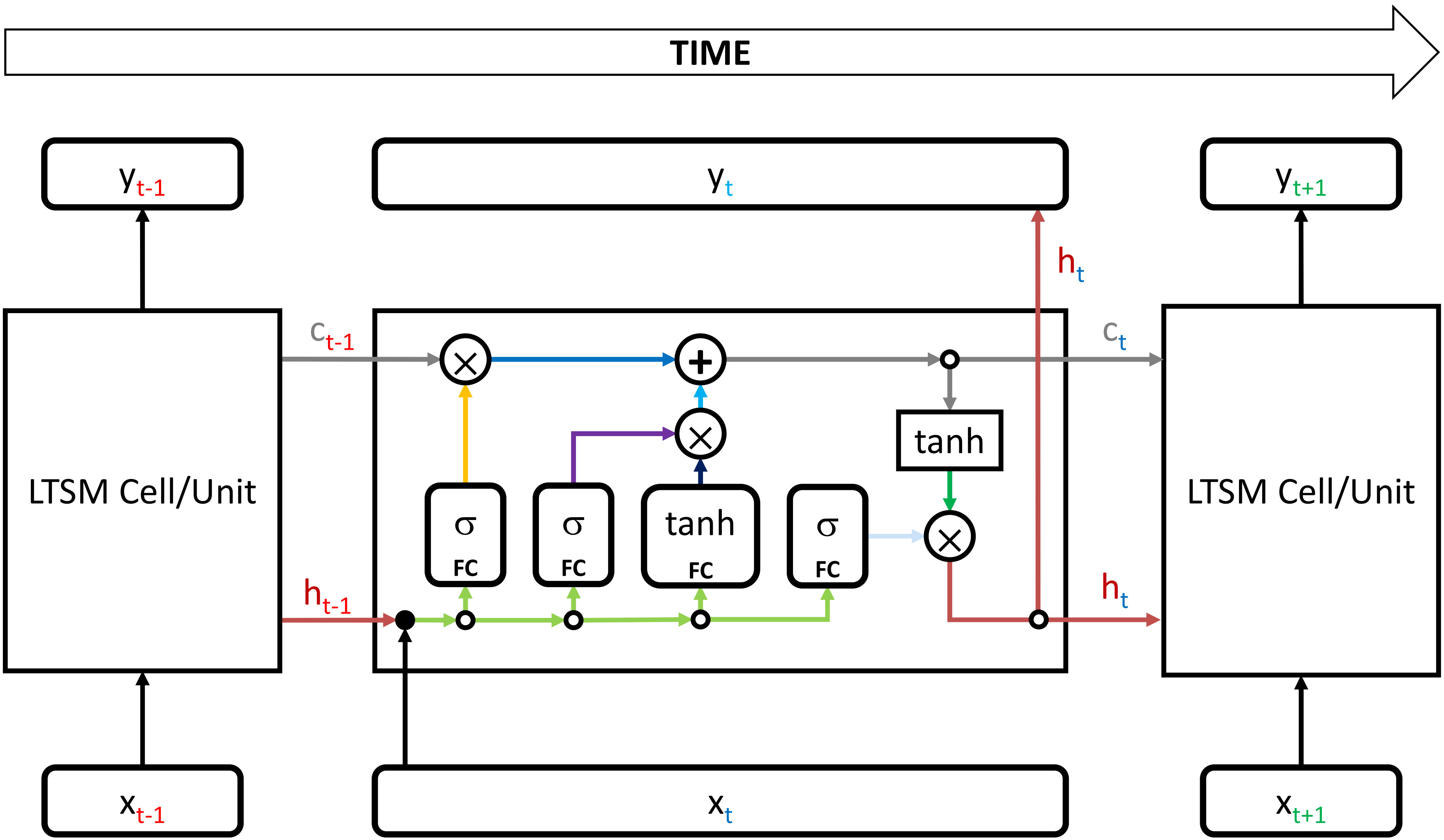
Long Short Term Memory (LSTM)

Long Short Term Memory (LSTM)

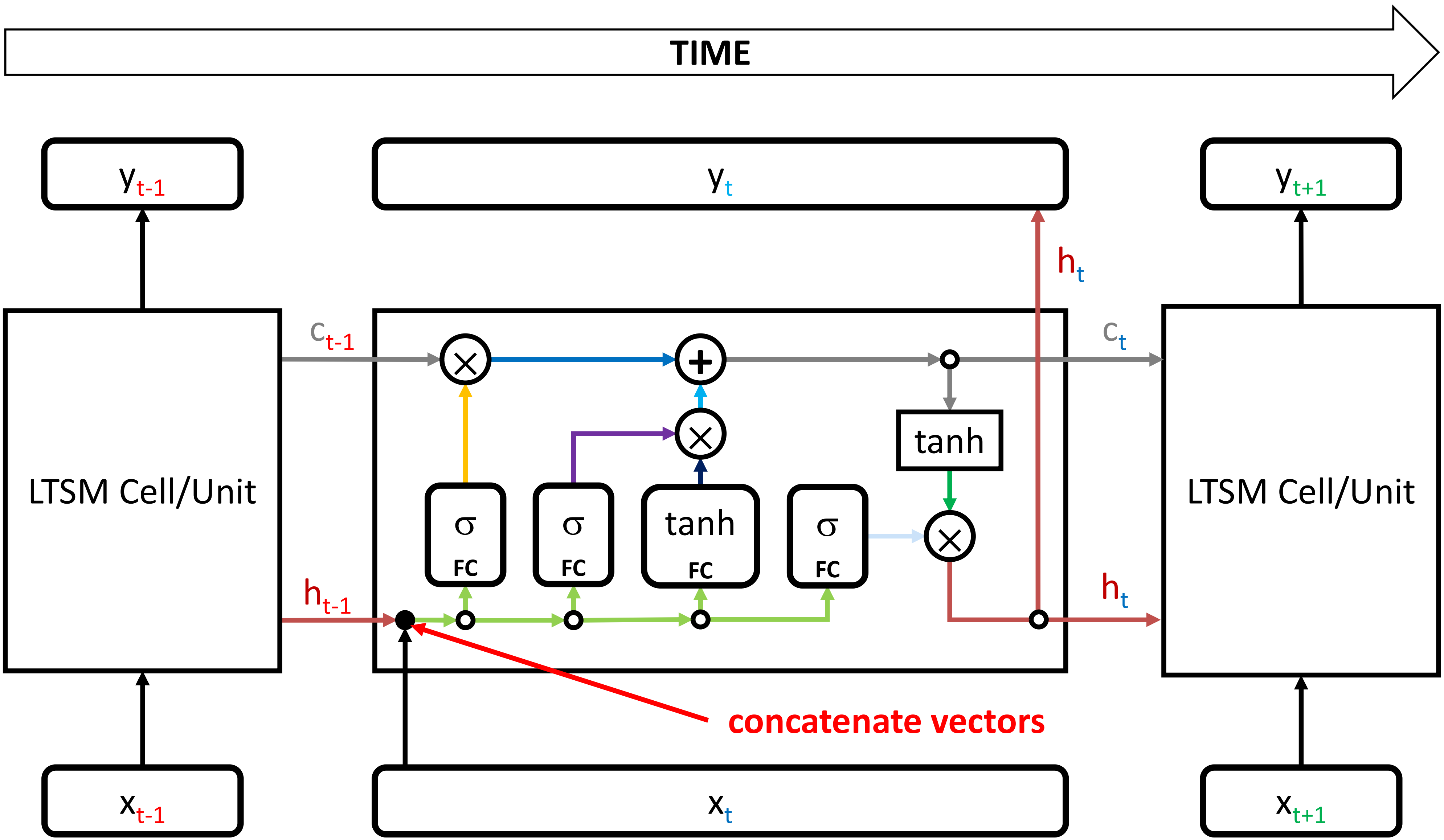
- A more sophisticated **version of the recurrent neural network** is the **Long Short Term Memory** (LSTM)

- An LSTM **uses gates** to **determine what information feeds forward** from one time step of the network to the next
  - this **helps to address the vanishing/exploding gradient** problems and make learning more stable

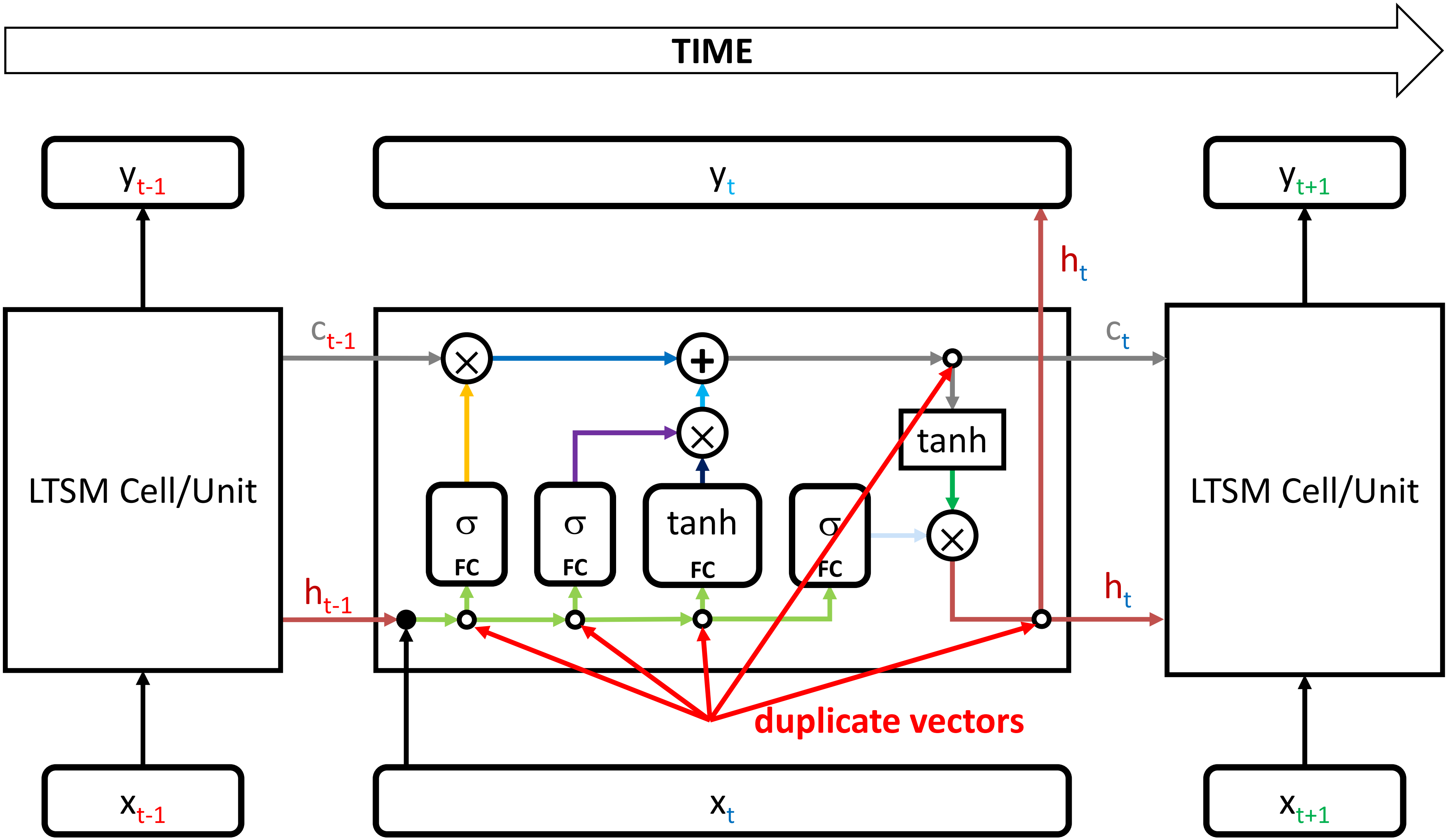
LSTM Cell/Unit



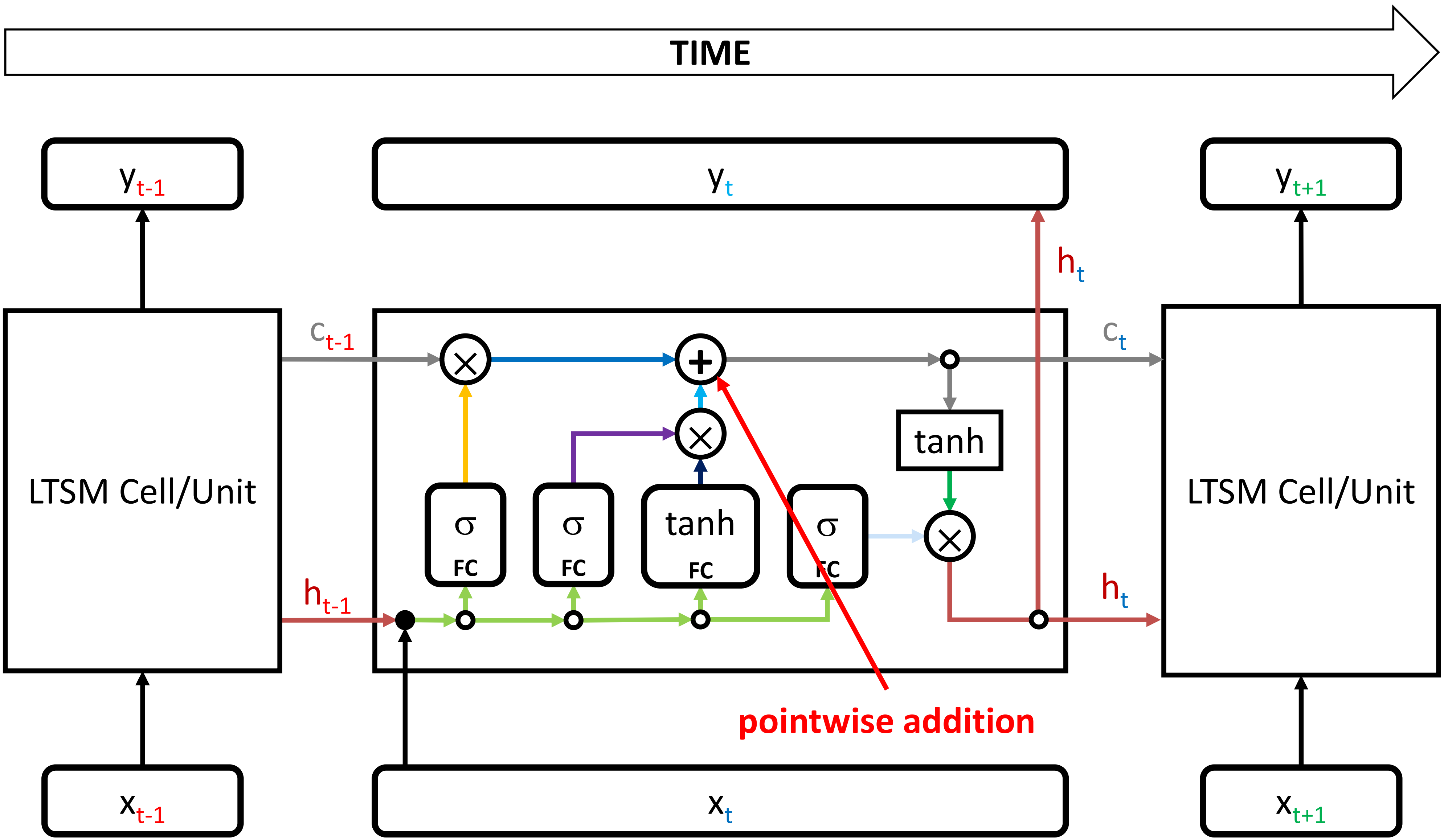
LSTM Cell/Unit



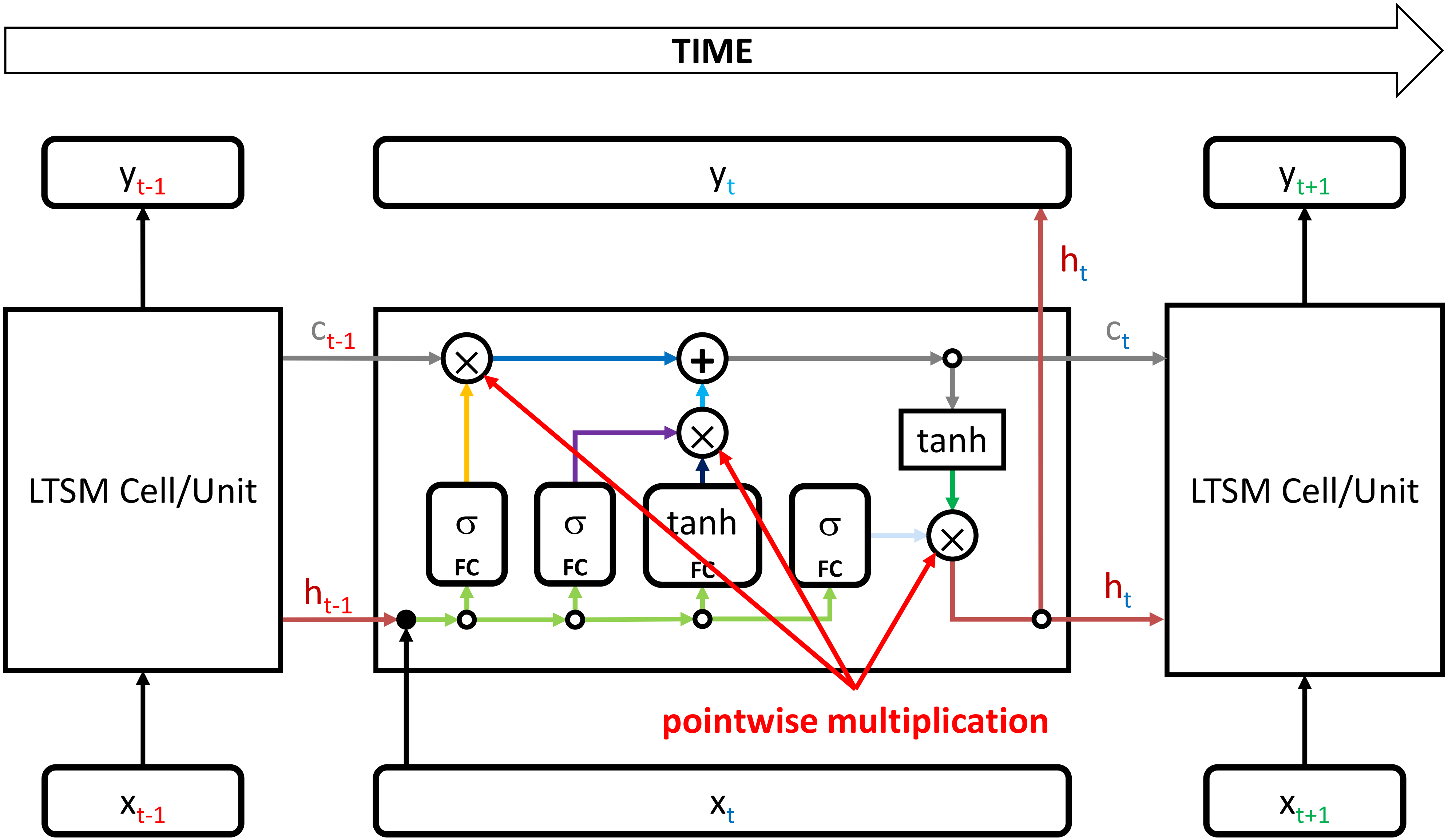
LSTM Cell/Unit



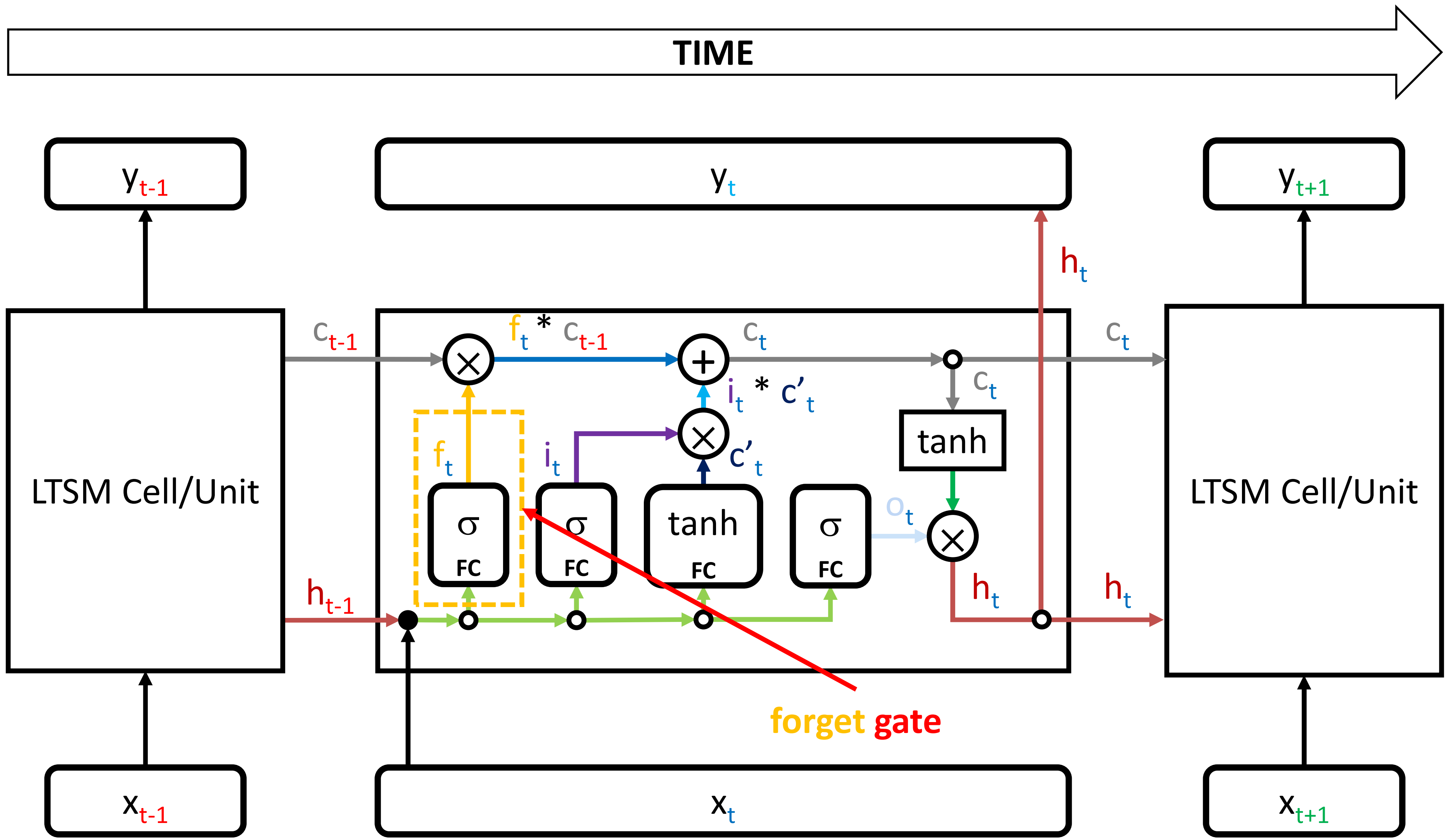
LSTM Cell/Unit



LSTM Cell/Unit

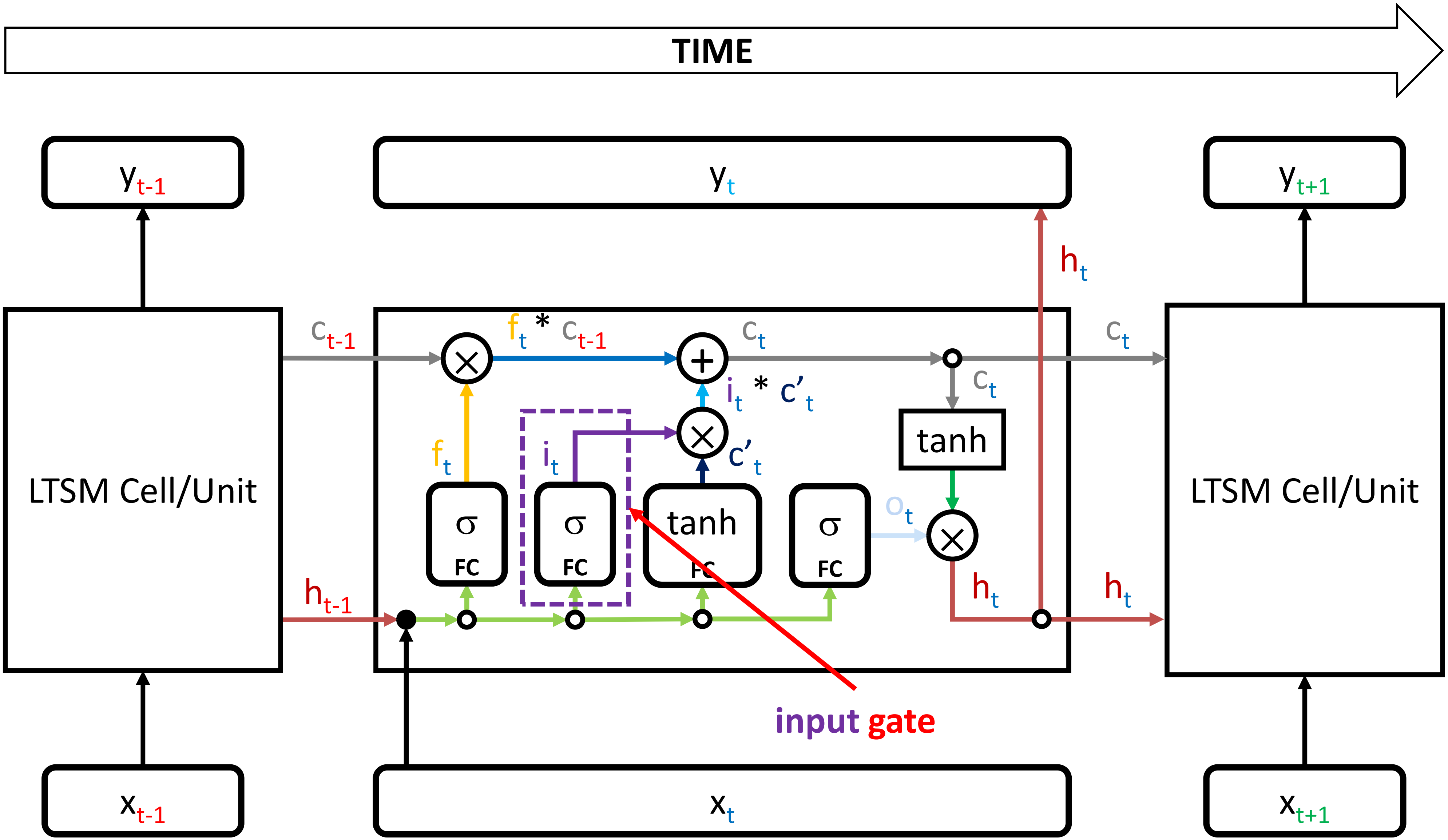


LSTM Cell/Unit



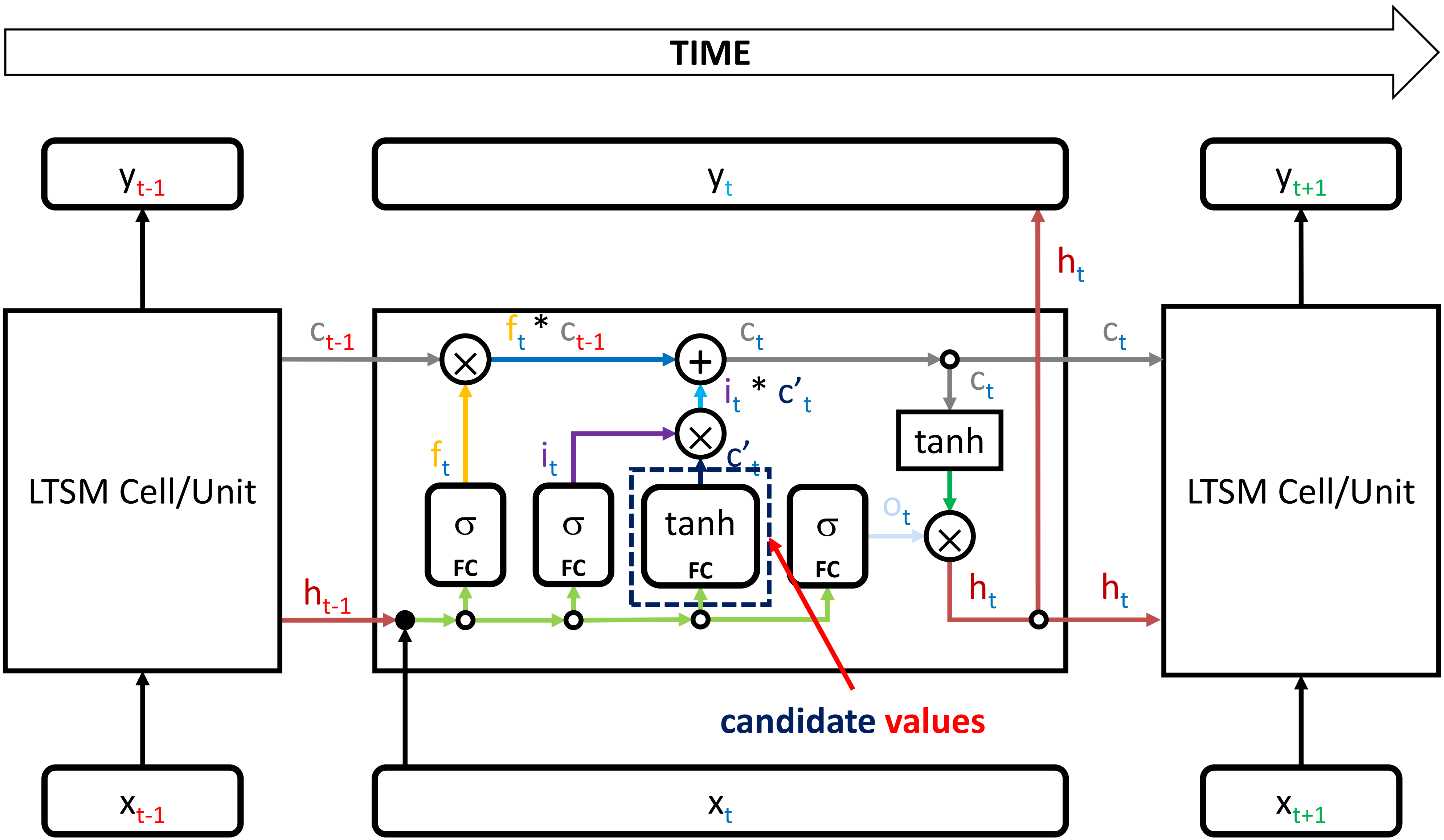
forget gate: determines how much of previous cell state is incorporated into current cell state

LSTM Cell/Unit

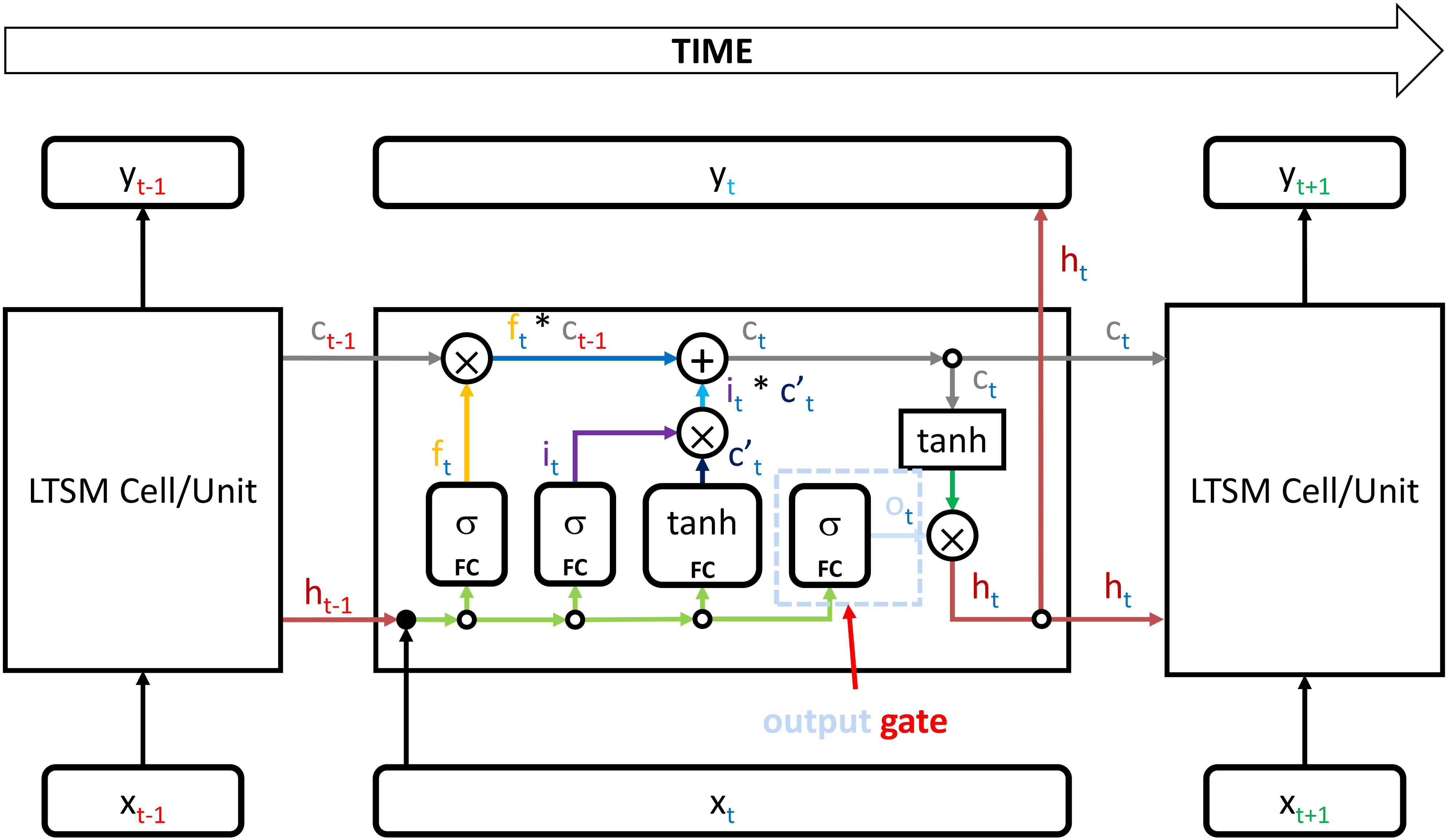


input gate: determines how much of input is incorporated into cell state

LSTM Cell/Unit

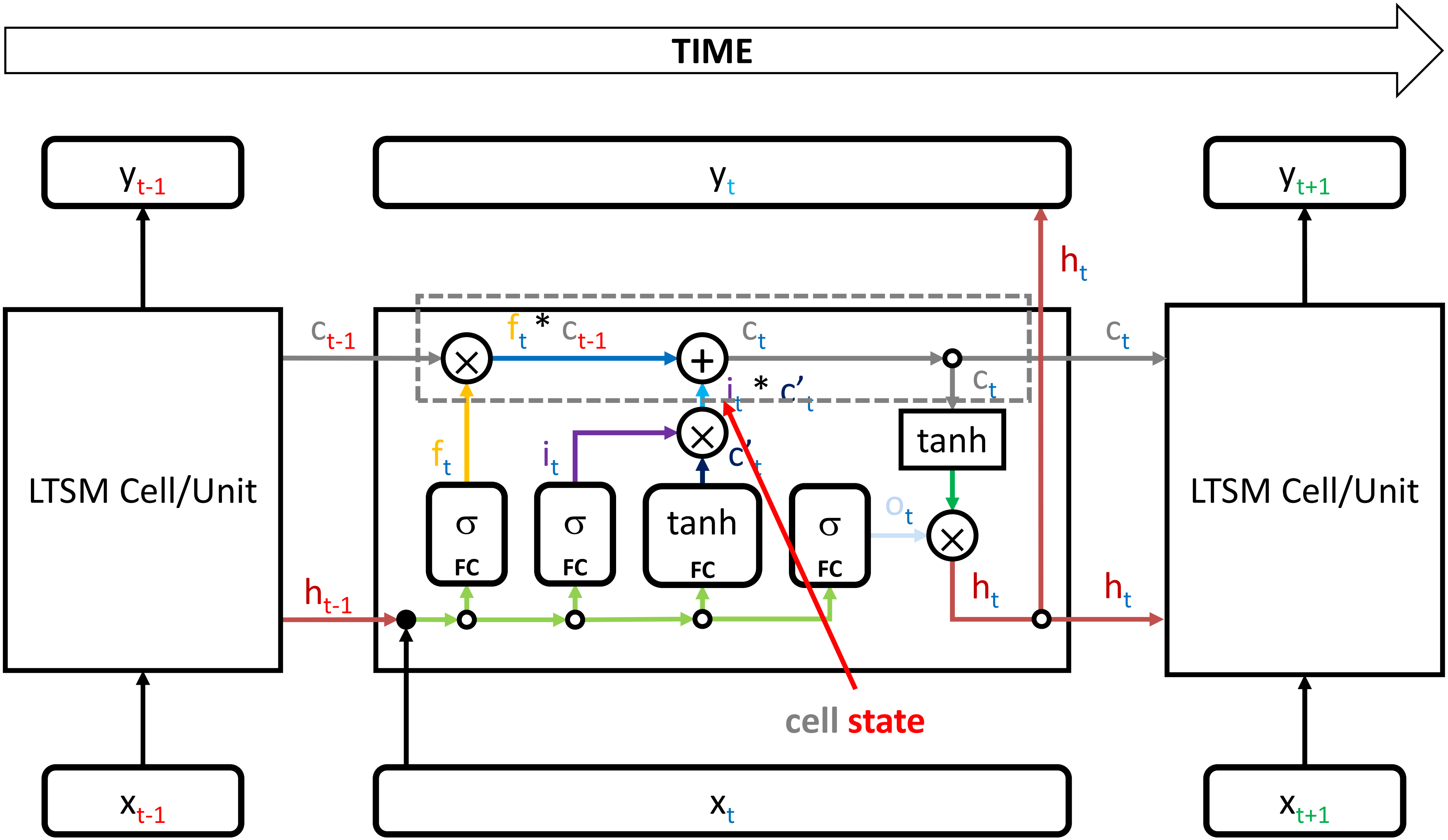


LSTM Cell/Unit

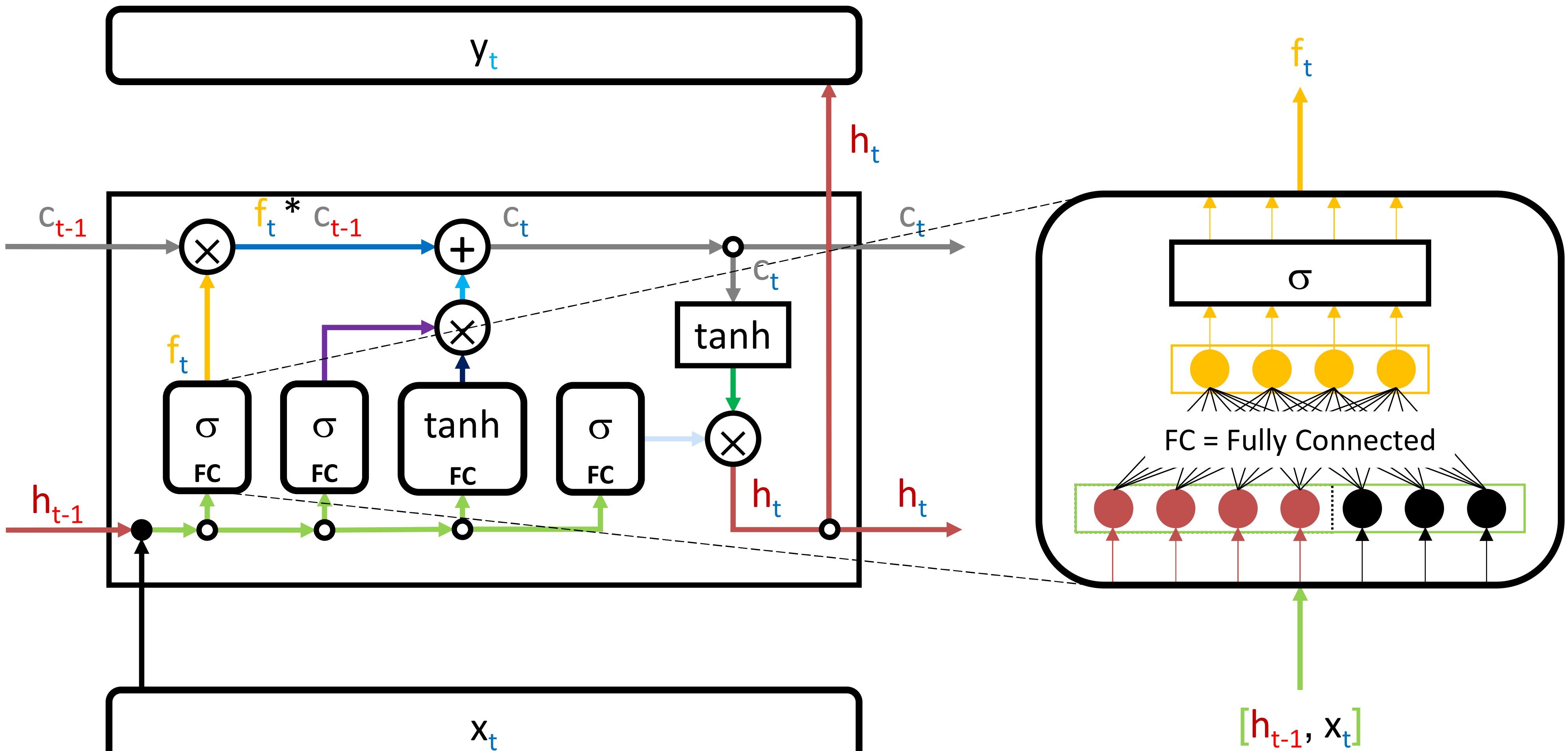


output gate: determines how much of current cell state is incorporated into current output

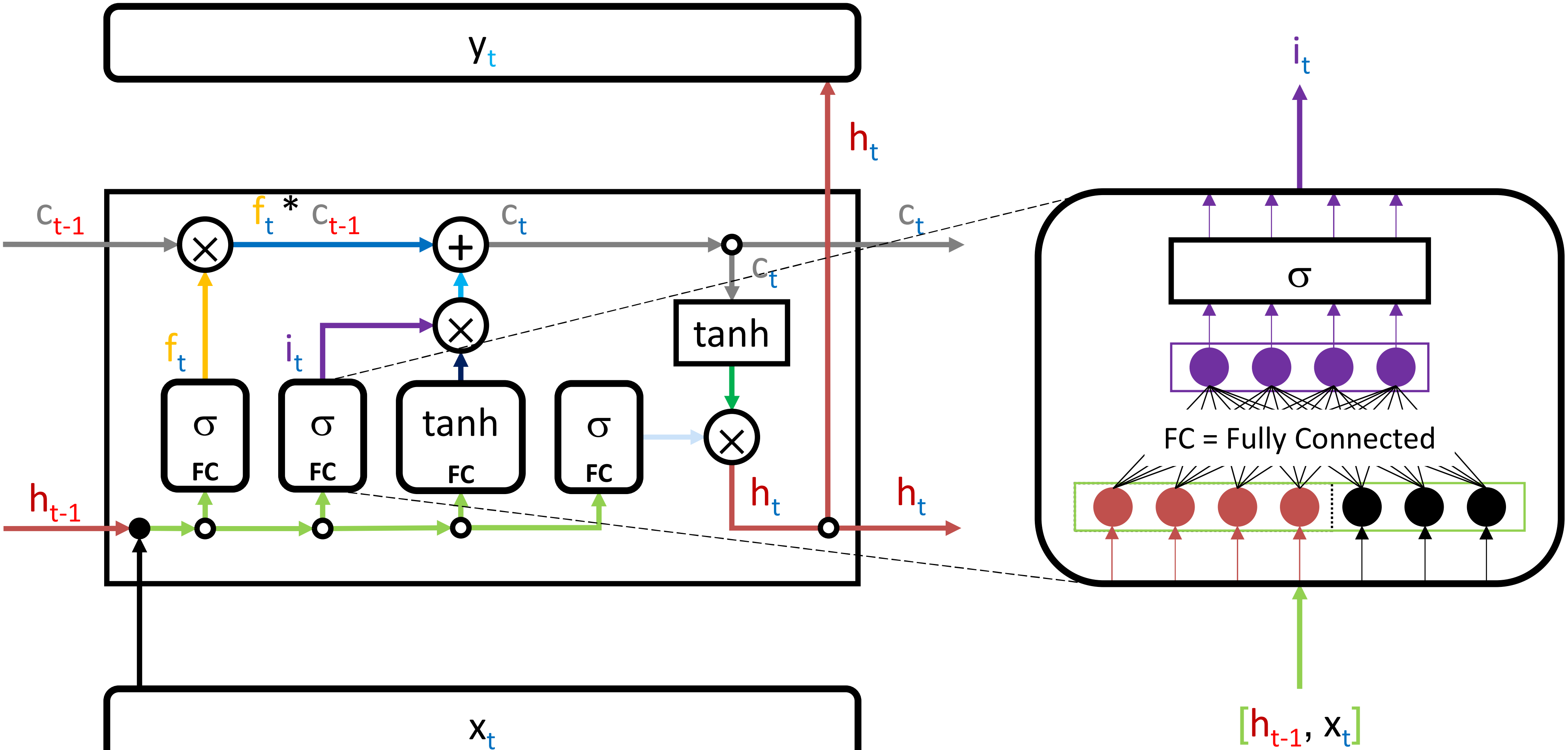
LSTM Cell/Unit



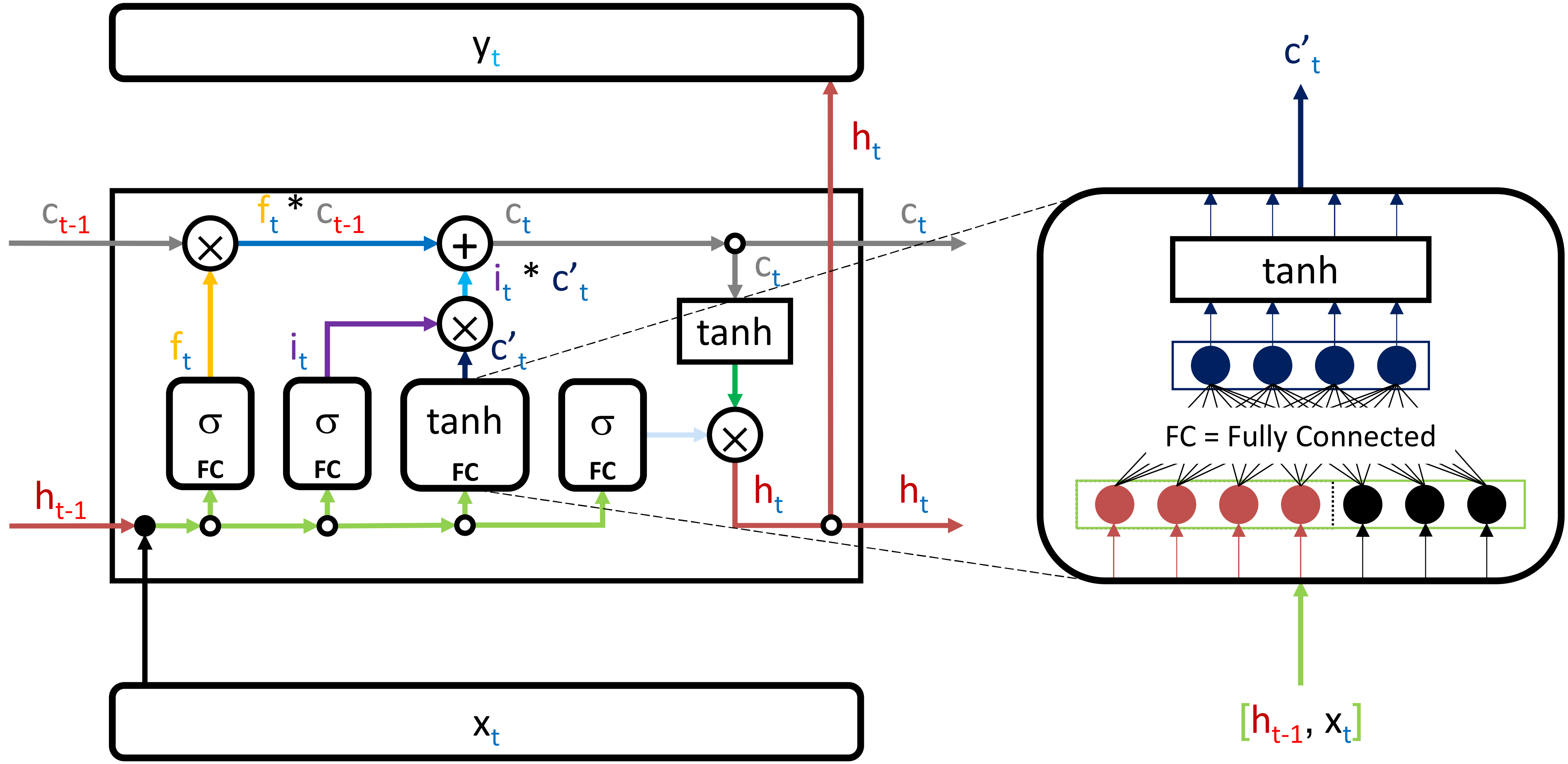
LSTM Cell/Unit



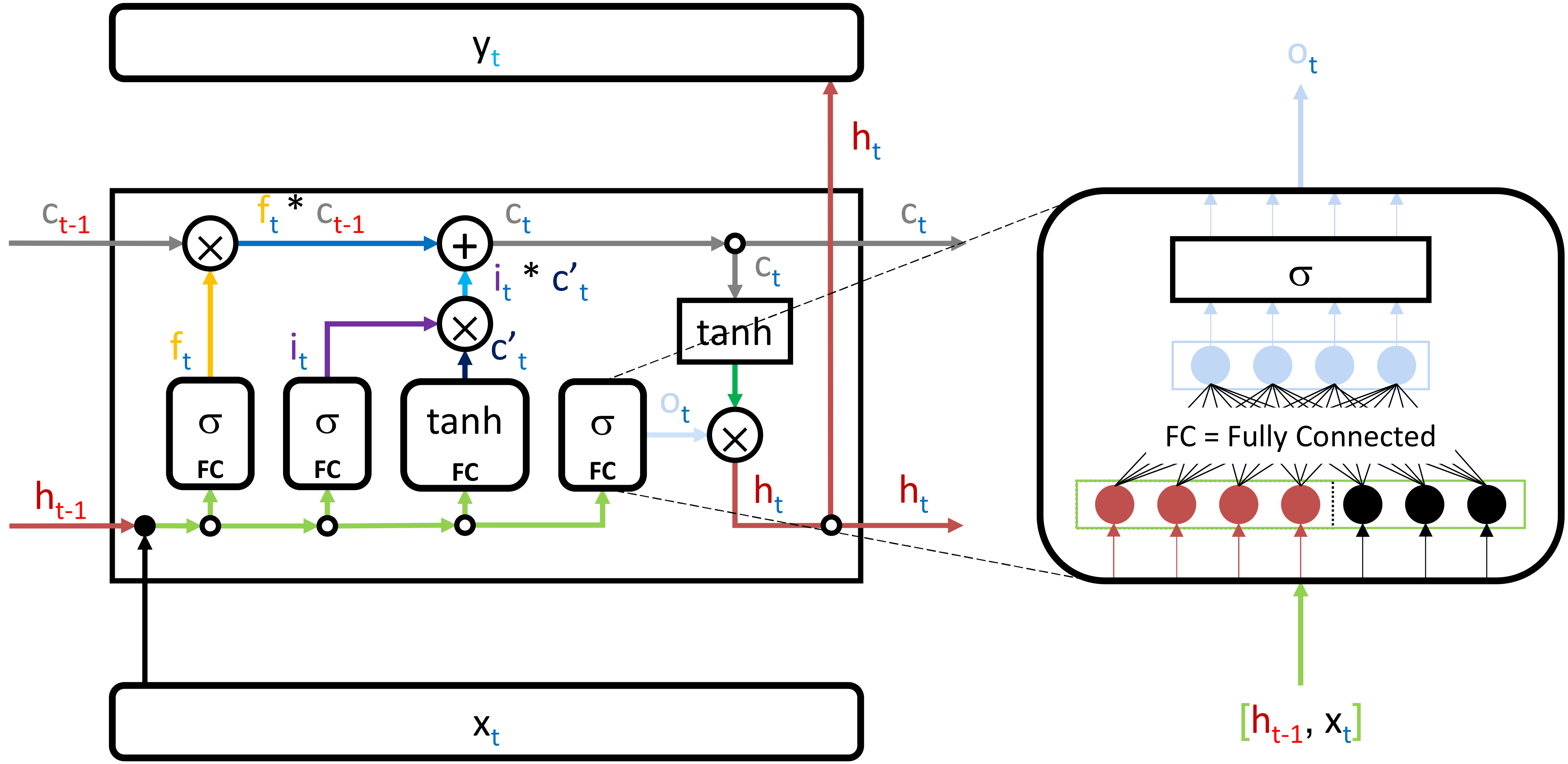
LSTM Cell/Unit



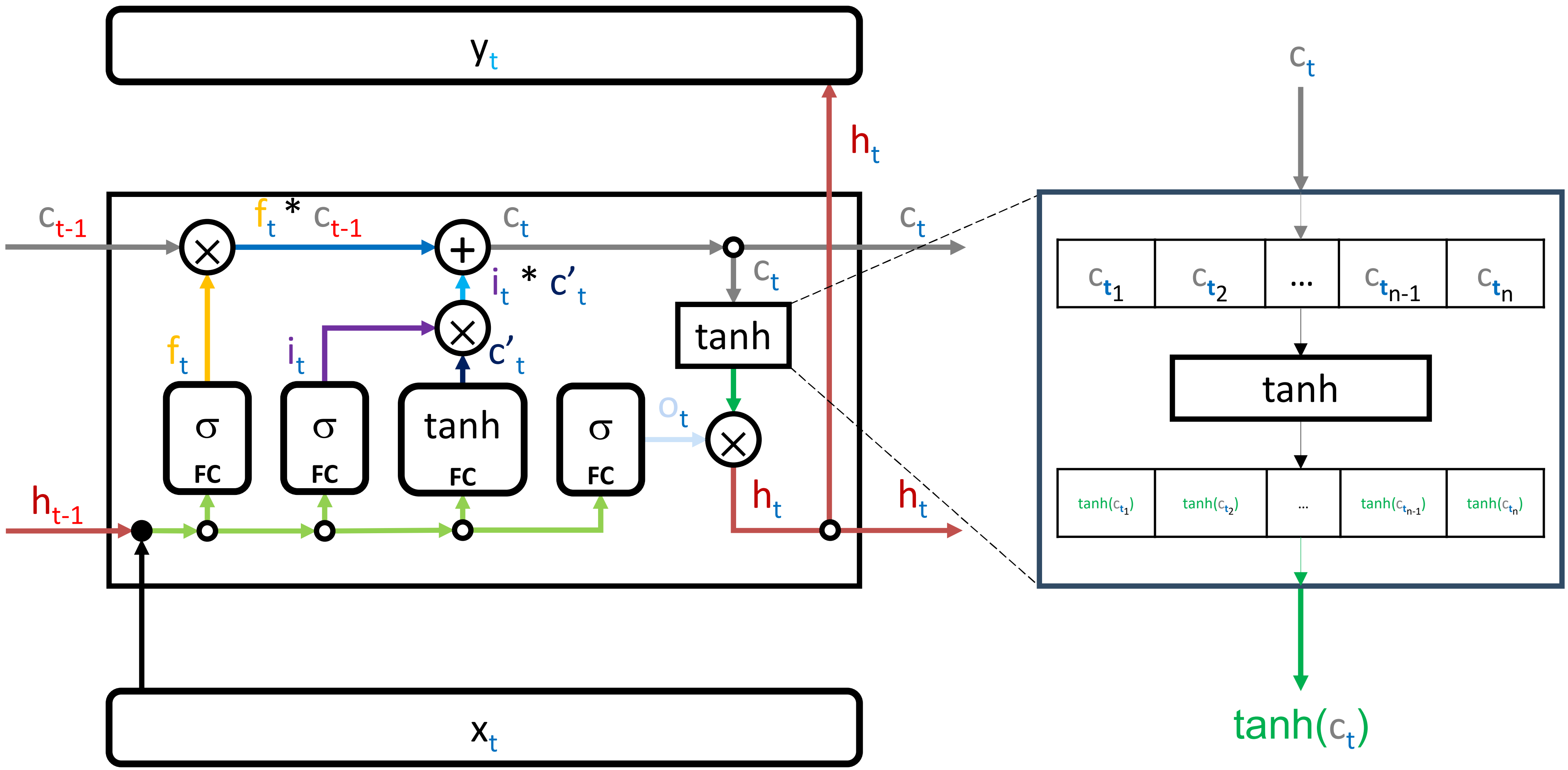
LSTM Cell/Unit



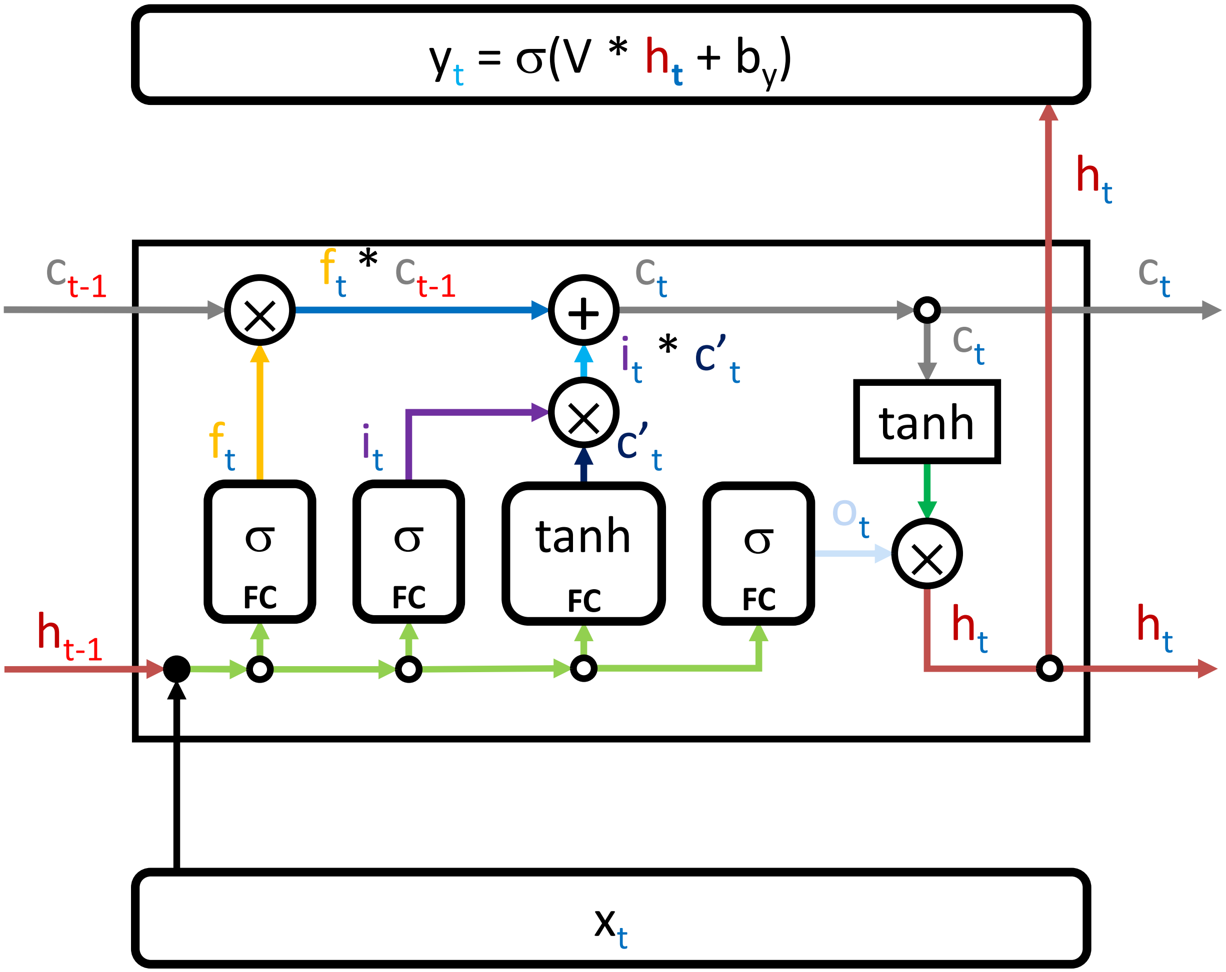
LSTM Cell/Unit



LSTM Cell/Unit

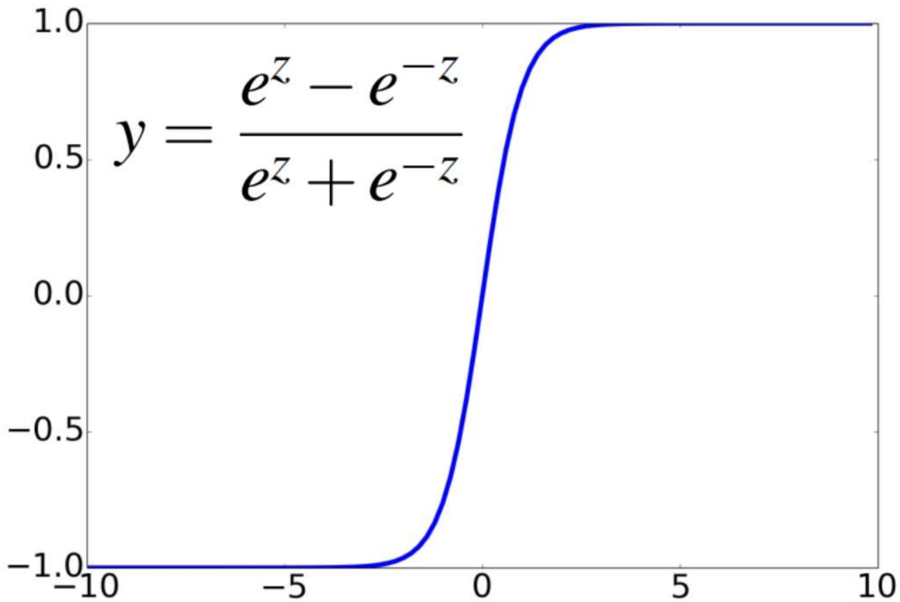


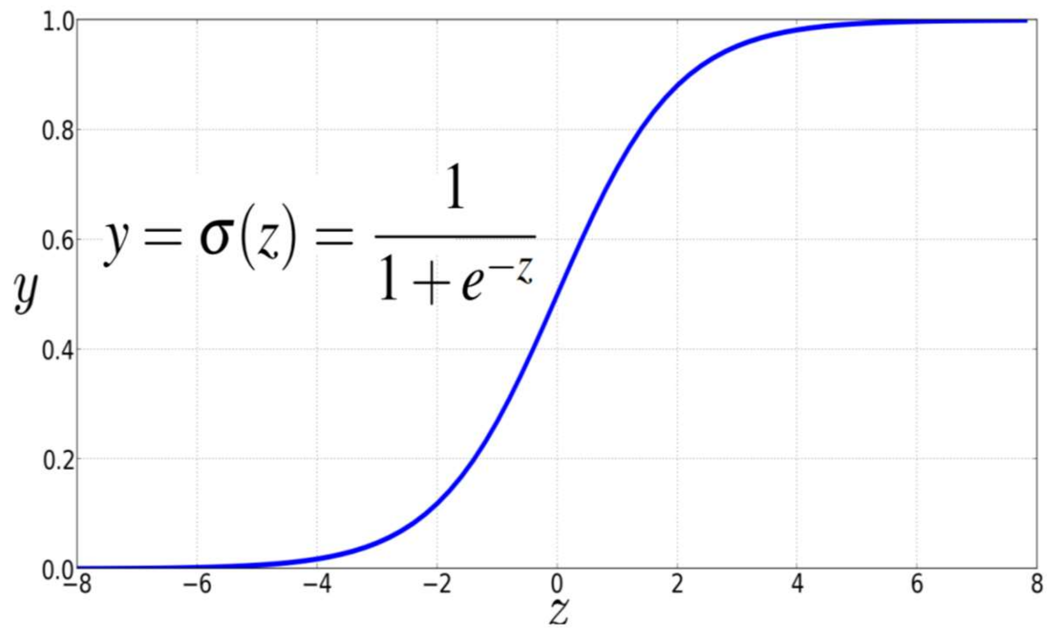
LSTM Cell/Unit



f_t = forget gate output
i_t = input gate output
c'_t = candidate values
o_t = output gate value
h_t = new hidden state
c_t = new cell state

tanh and sigmoid Activation Functions

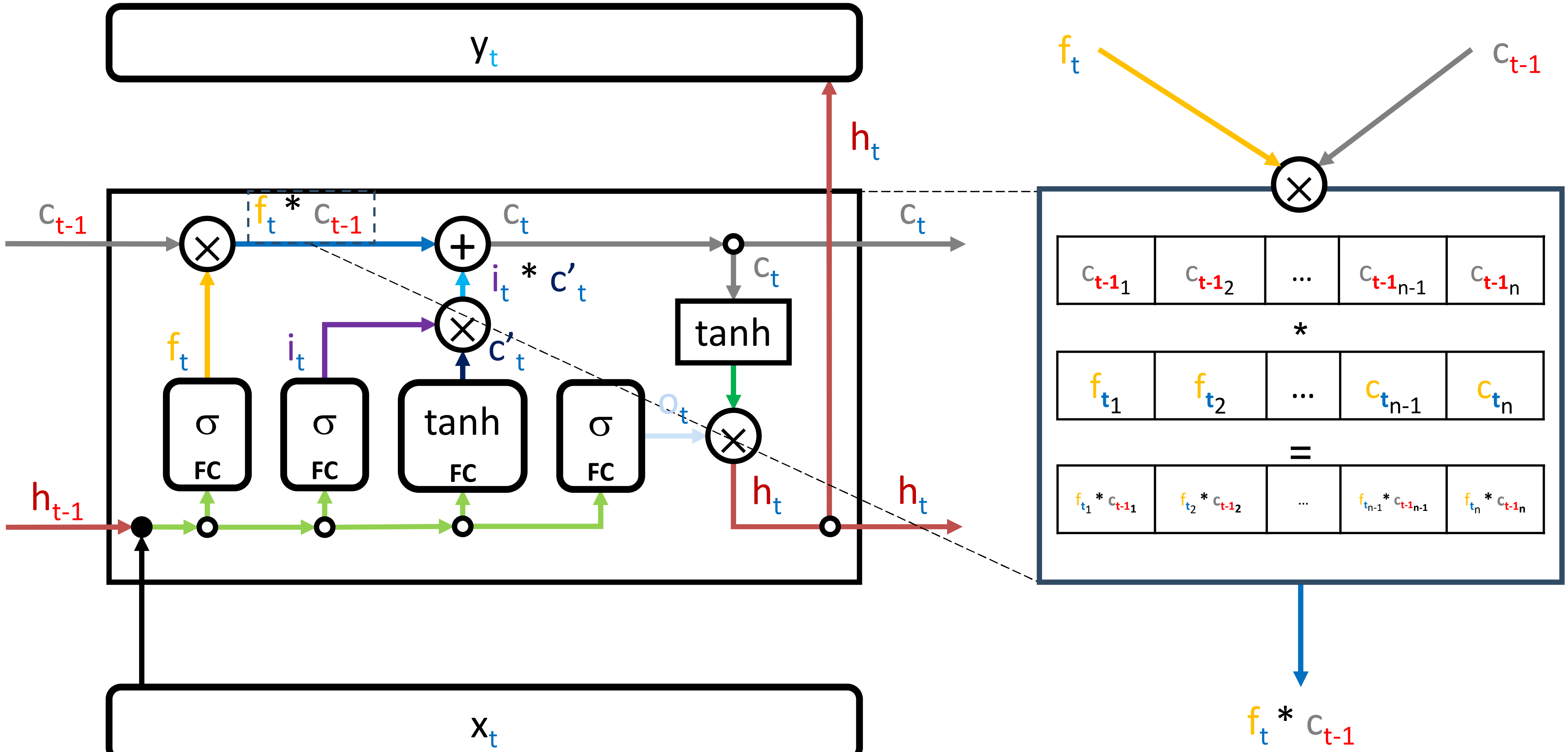




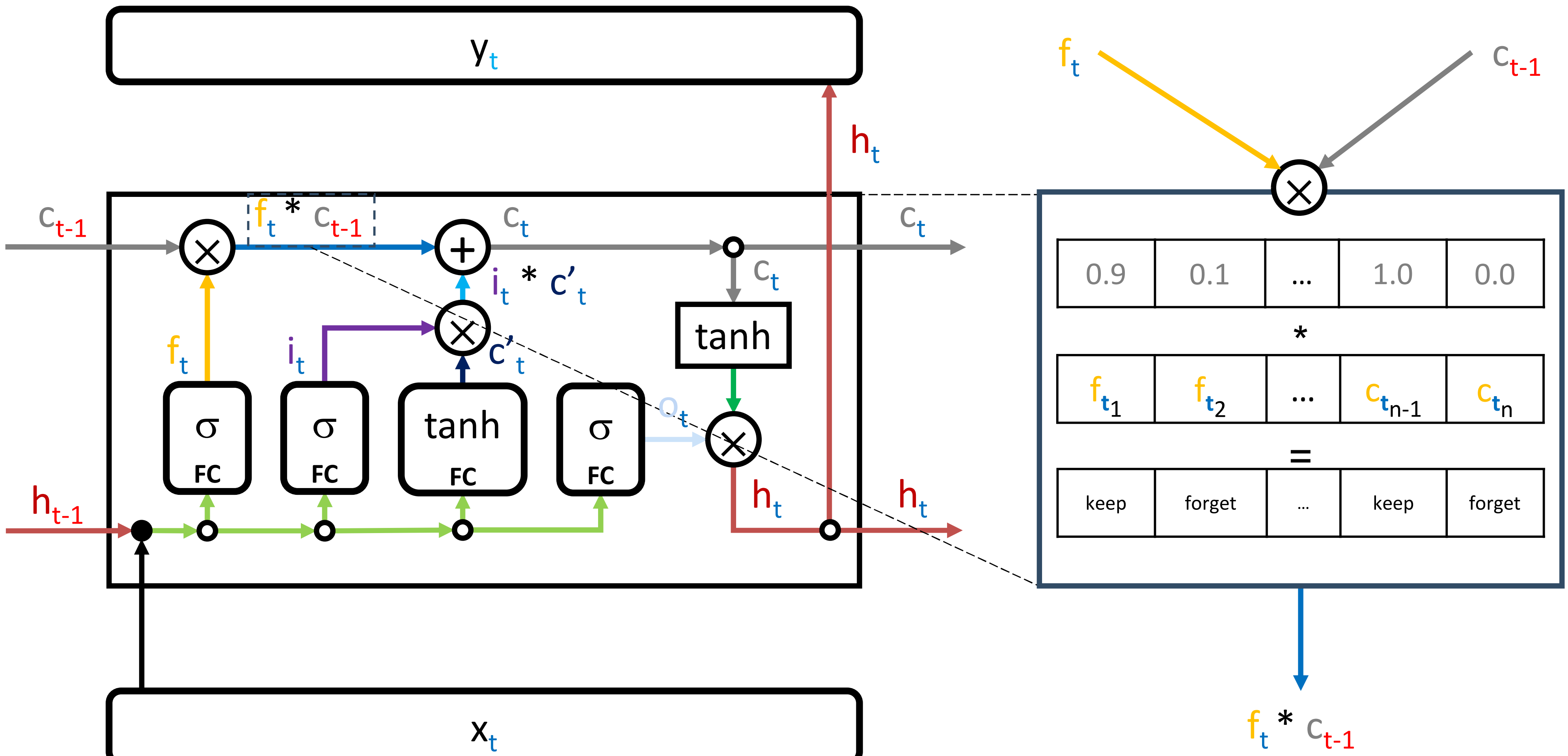
tanh
→ [-1,1] range

Sigmoid
→ [0,1] range

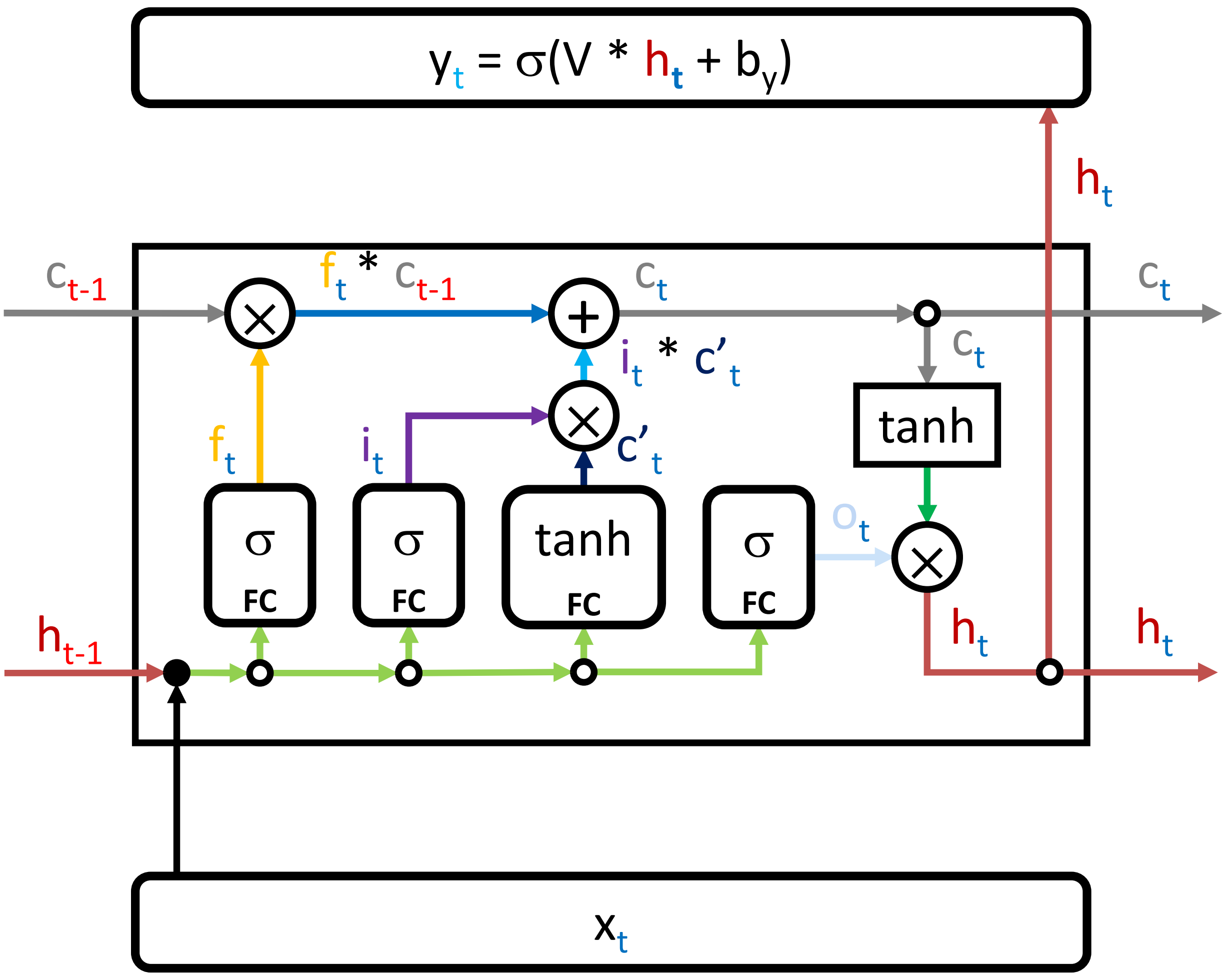
LSTM Cell/Unit



LSTM Cell/Unit



LSTM Cell/Unit



$f_t = \sigma(W_f * [h_{t-1}, x_t] + b_f)$

$i_t = \sigma(W_i * [h_{t-1}, x_t] + b_i)$

$c'_t = \tanh(W_c * [h_{t-1}, x_t] + b_c)$

$o_t = \sigma(W_o * [h_{t-1}, x_t] + b_i)$

$h_t = o_t * \tanh(c_t)$

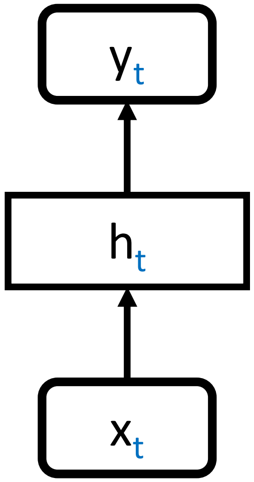
$c_t = f_t * c_{t-1} + i_t * c'_t$

biases (b_f, b_i, b_c, b_i, b_y) not shown on diagrams

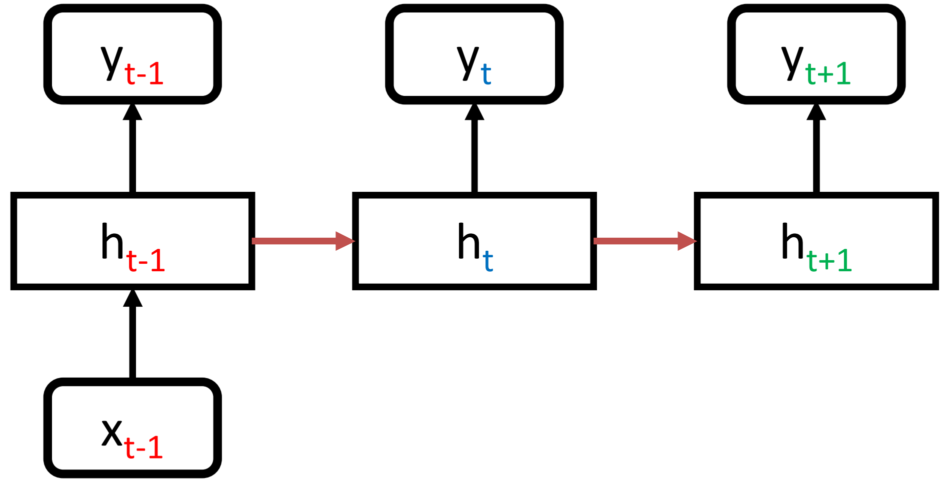
W_f, W_i, W_c, W_i, V are neural network weight matrices

RNN/LSTM Structure Types

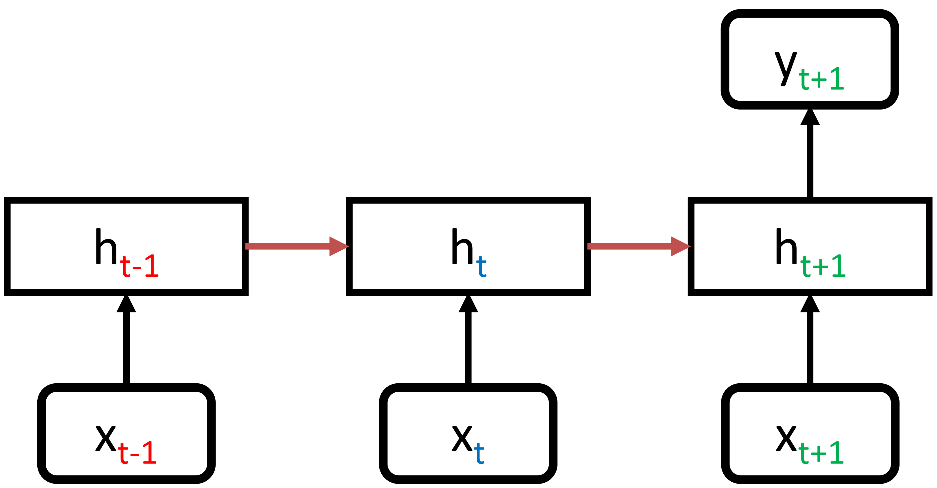
One to One



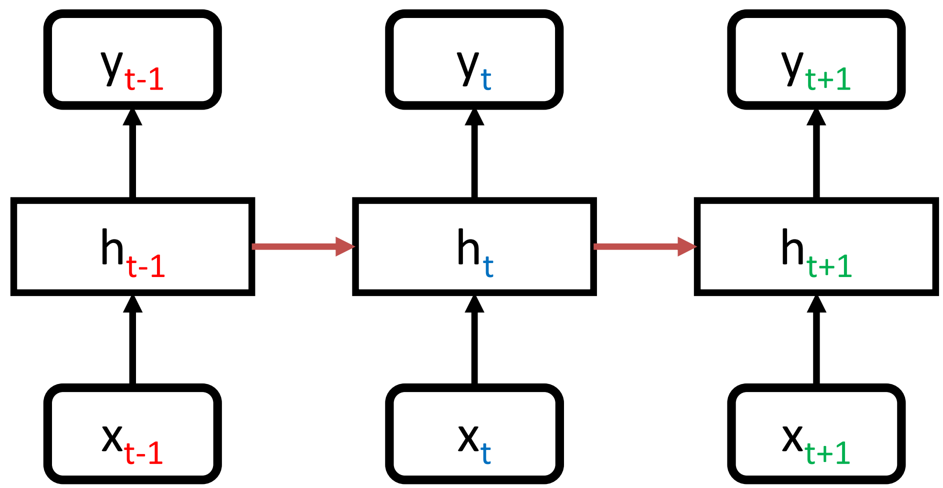
One to Many



Many to One



Many to Many



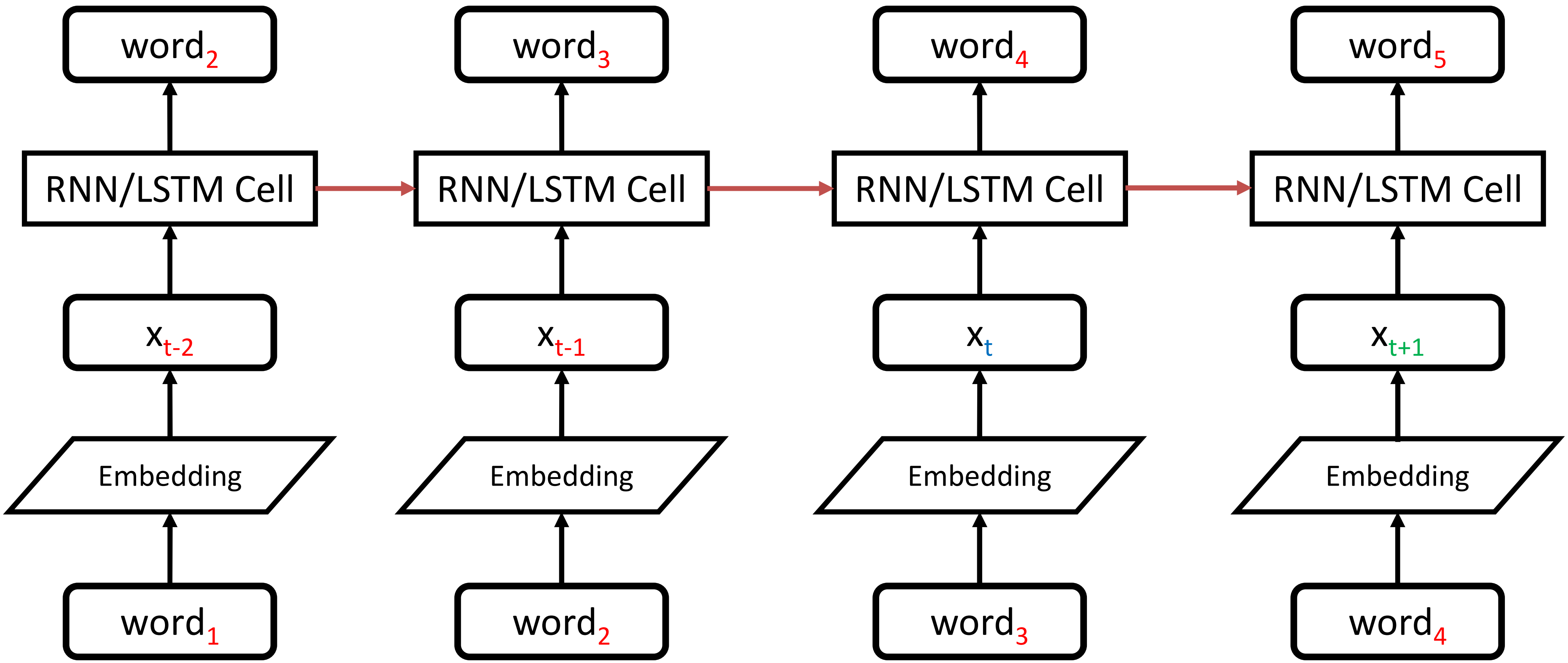
CNNs for Text Classification/Prediction

We noted before that some text categorization tasks (like sentiment analysis) could also benefit from using sequential information about the words in a text

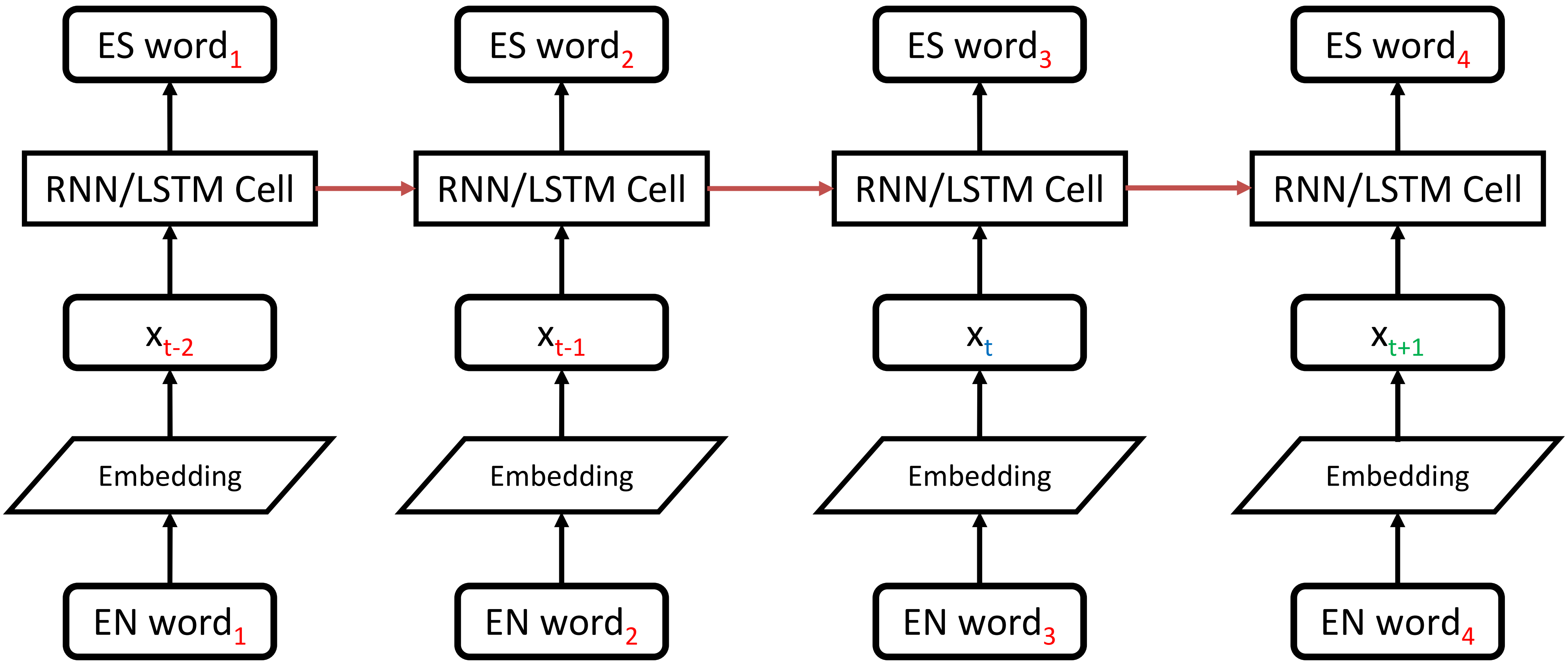
I would never buy this product again. It clearly failed under high-stress testing in my home.

I would clearly buy this product again. It never failed under high-stress testing in my home.

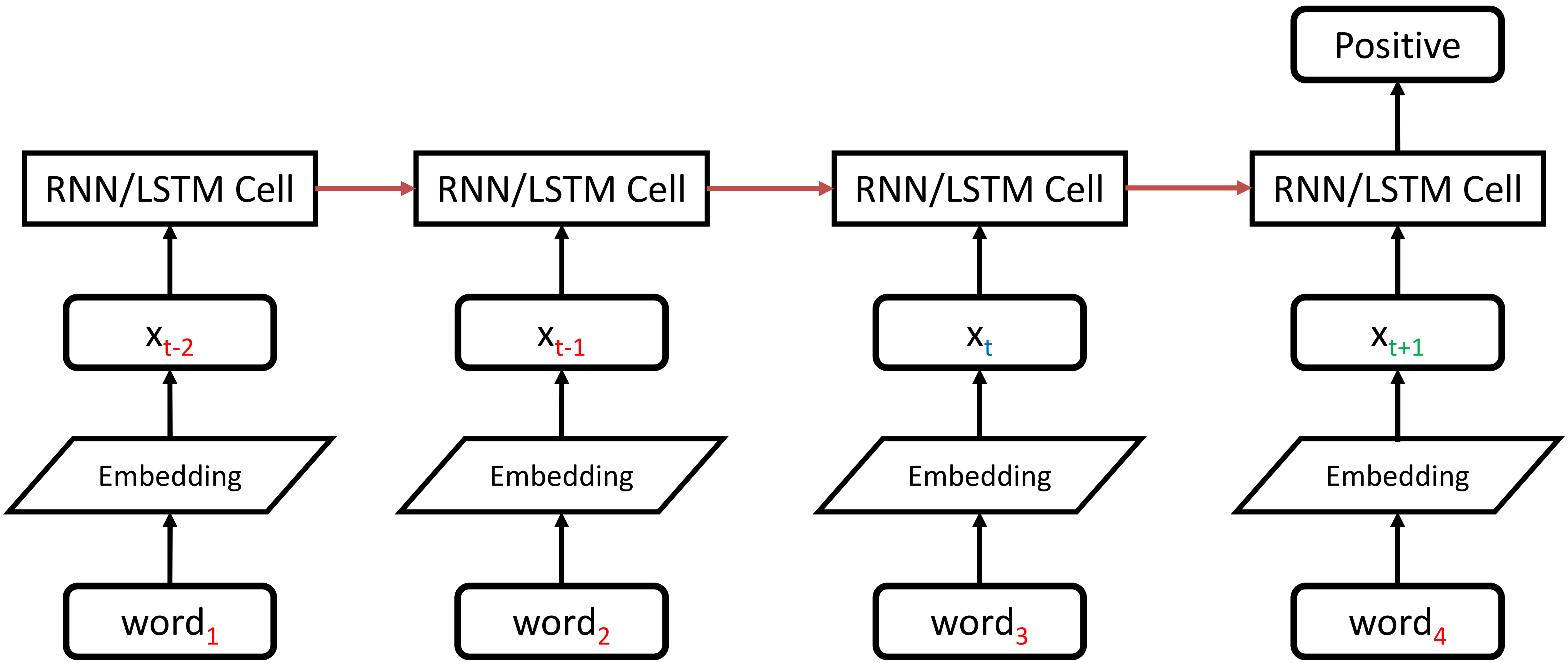
Many to Many: Word Prediction



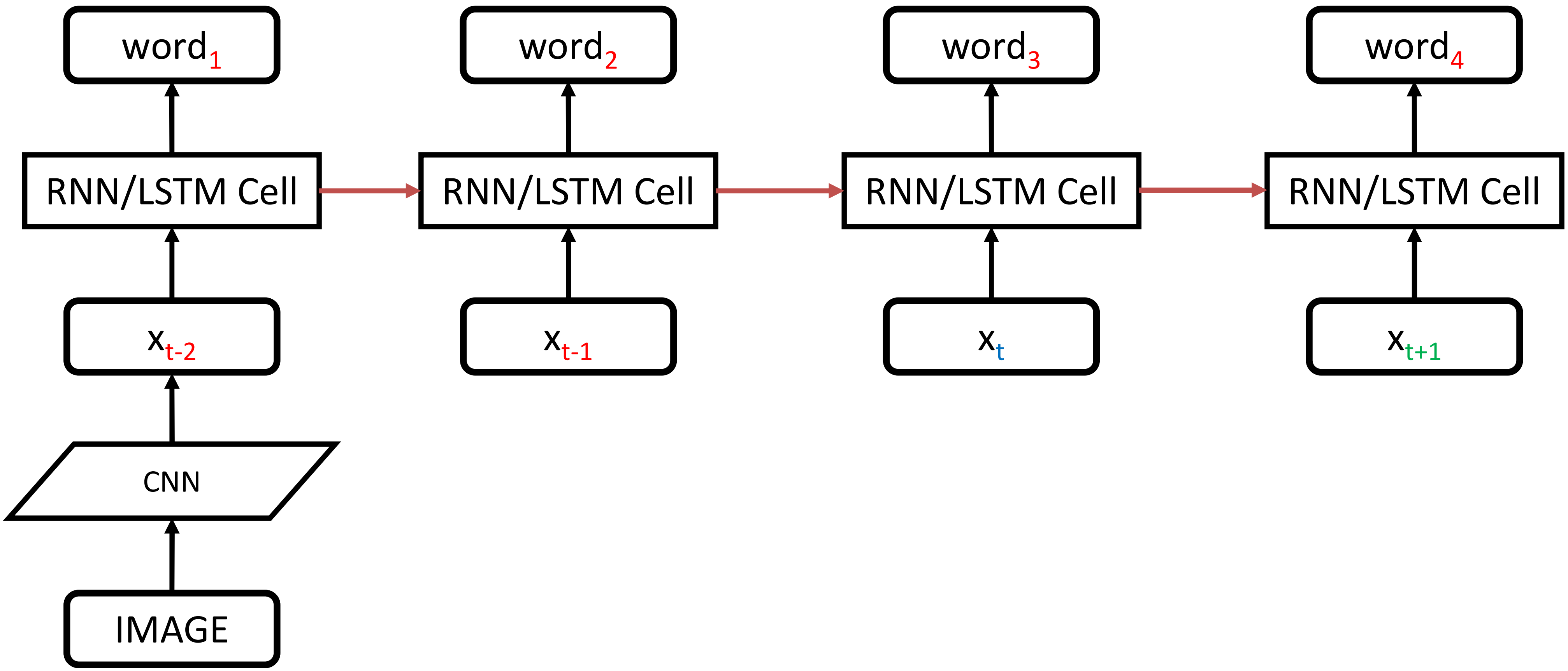
Many to Many: Translation



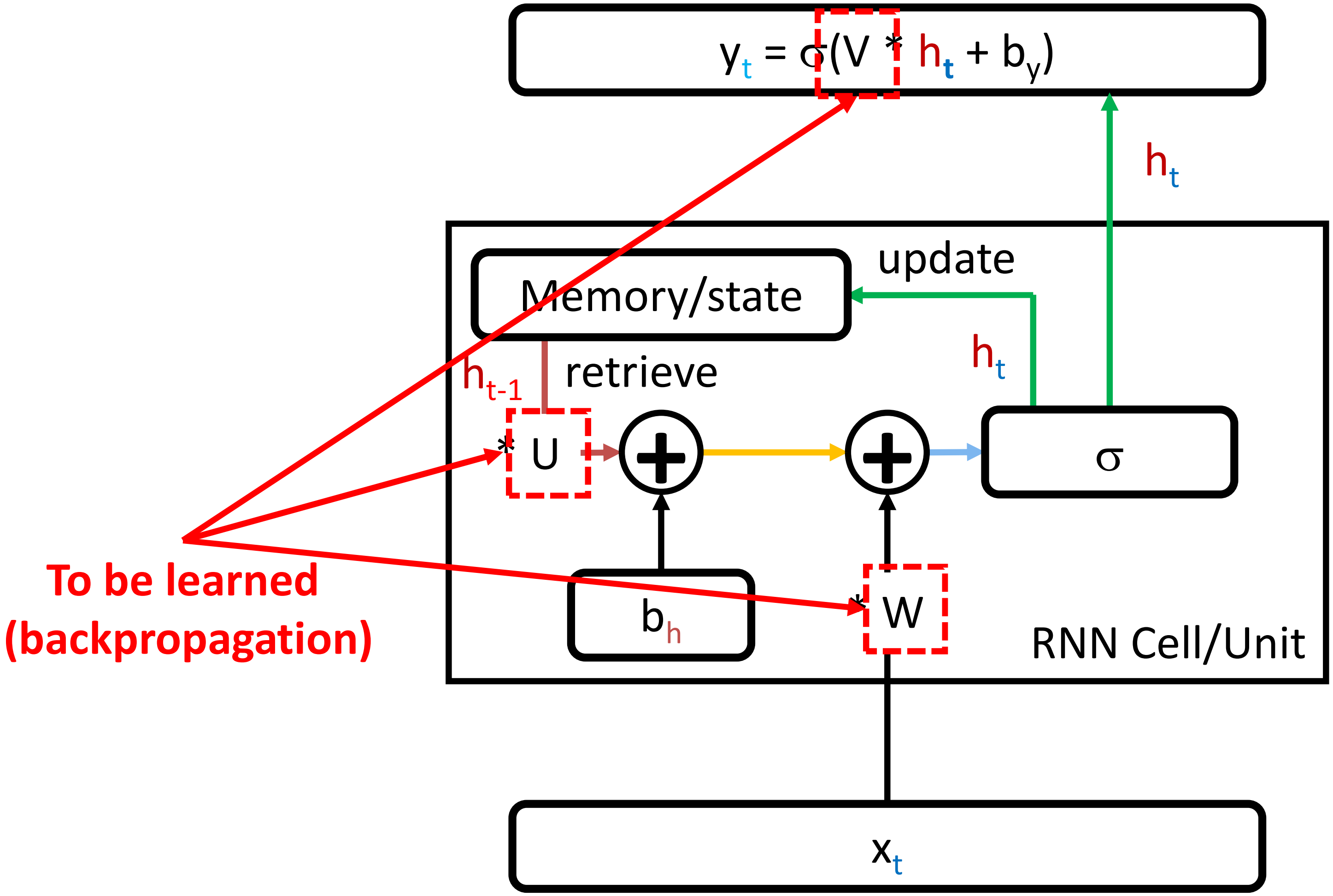
Many to One: Classification



One to Many: Image Captioning

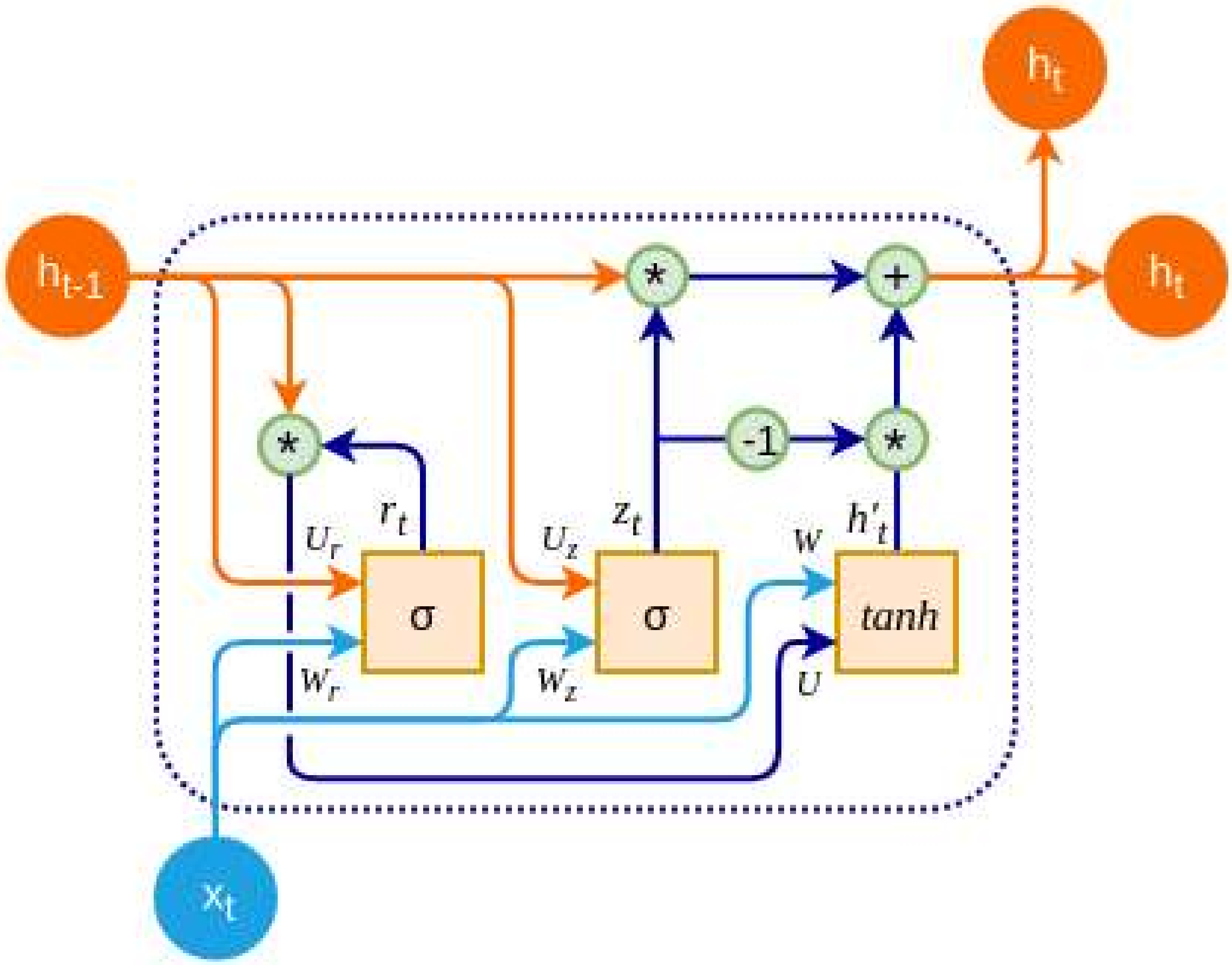


RNN Cell/Unit



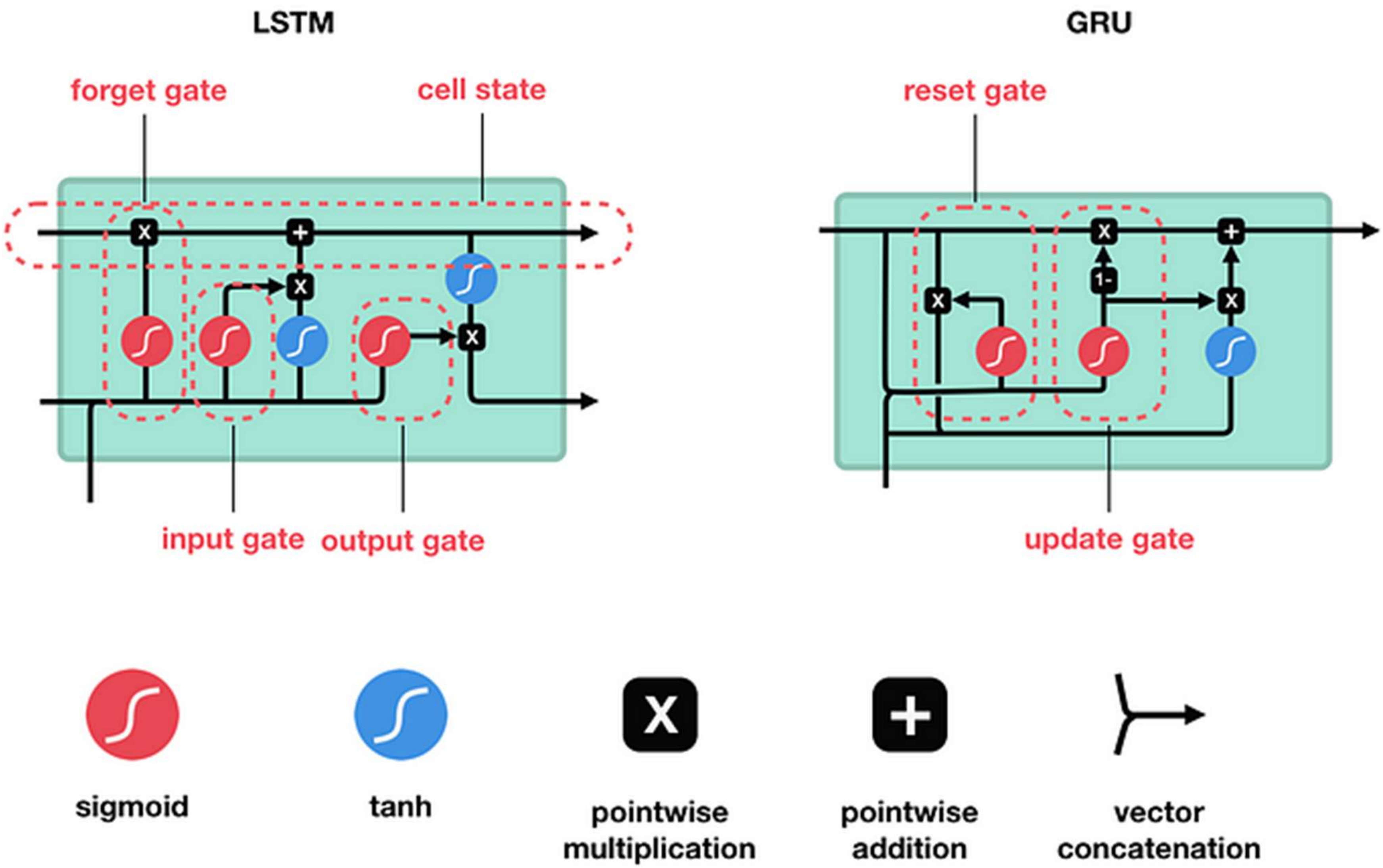
$h_t = \sigma(W * x_t + U * h_{t-1} + b_h)$

Gated Recurrent Unit (GRU)



source: https://www.oreilly.com/library/view/advanced-deep-learning/9781789956177/8ad9dc41-3237-483e-8f6b-7e5f653dc693.xhtml

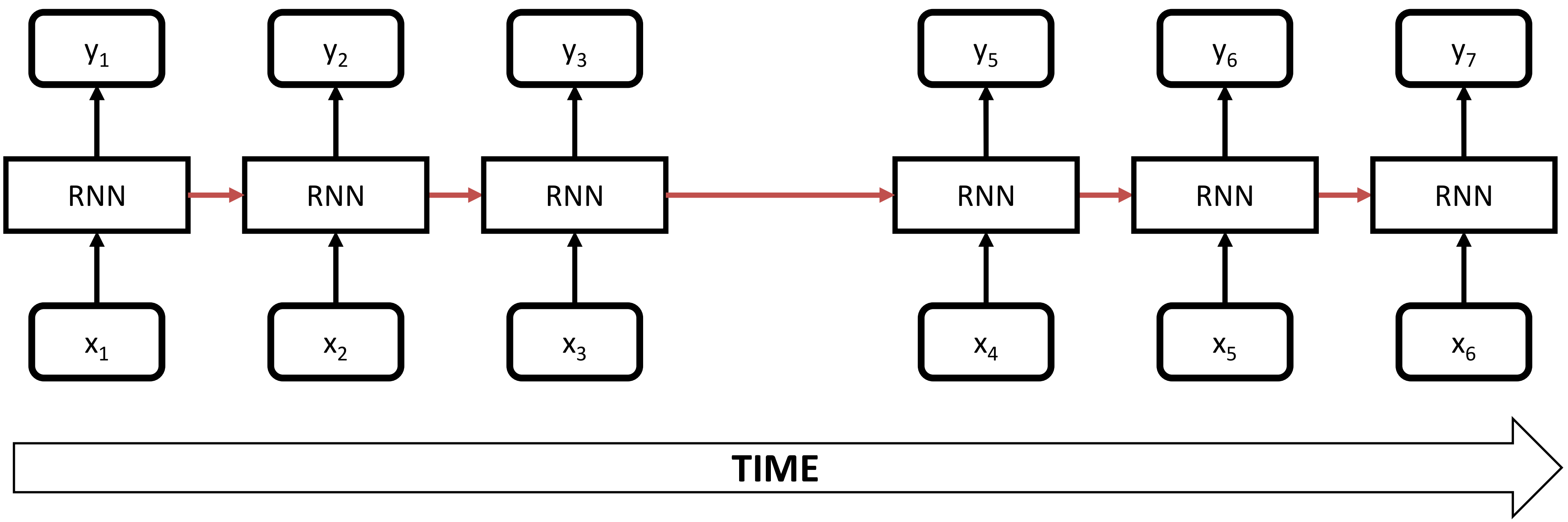
LSTM vs. Gated Recurrent Unit



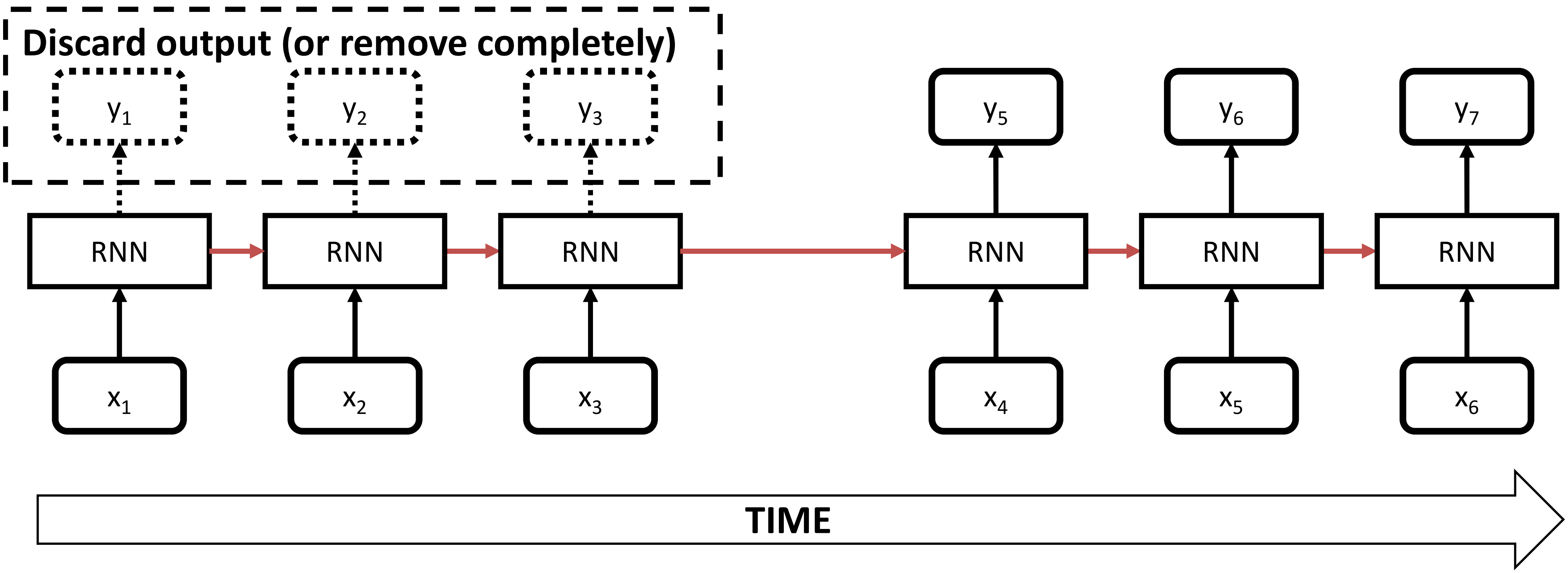
source: https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21

Sequence to Sequence Networks (seq2seq)

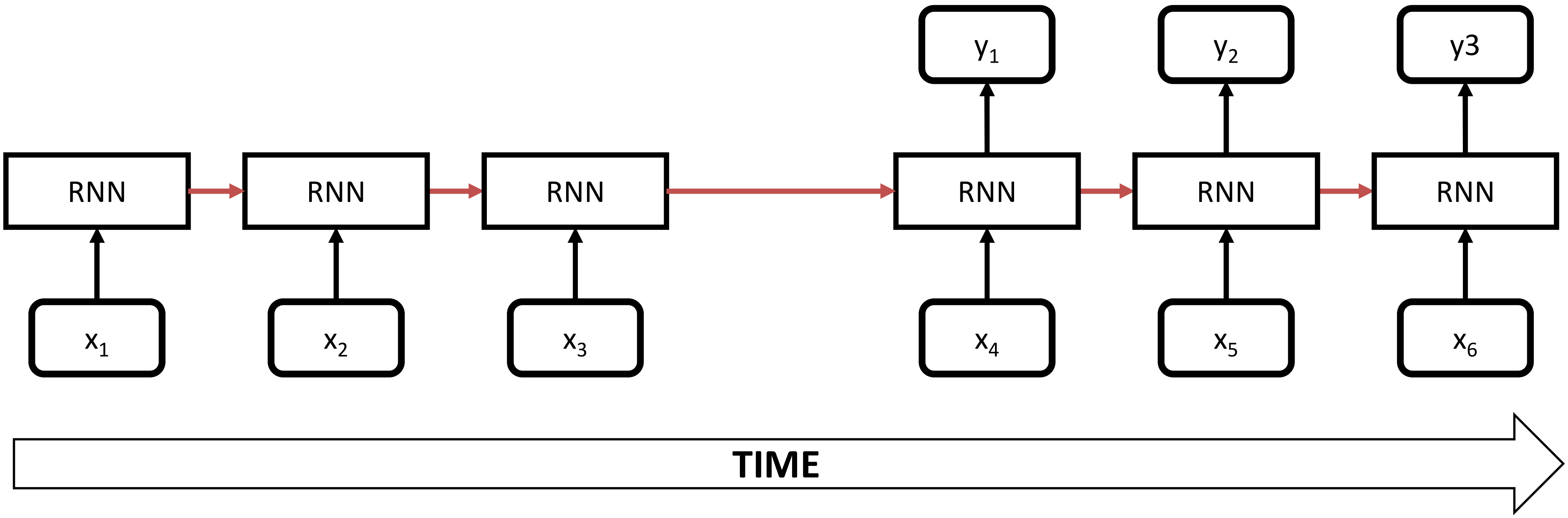
Many-to-Many RNN Network



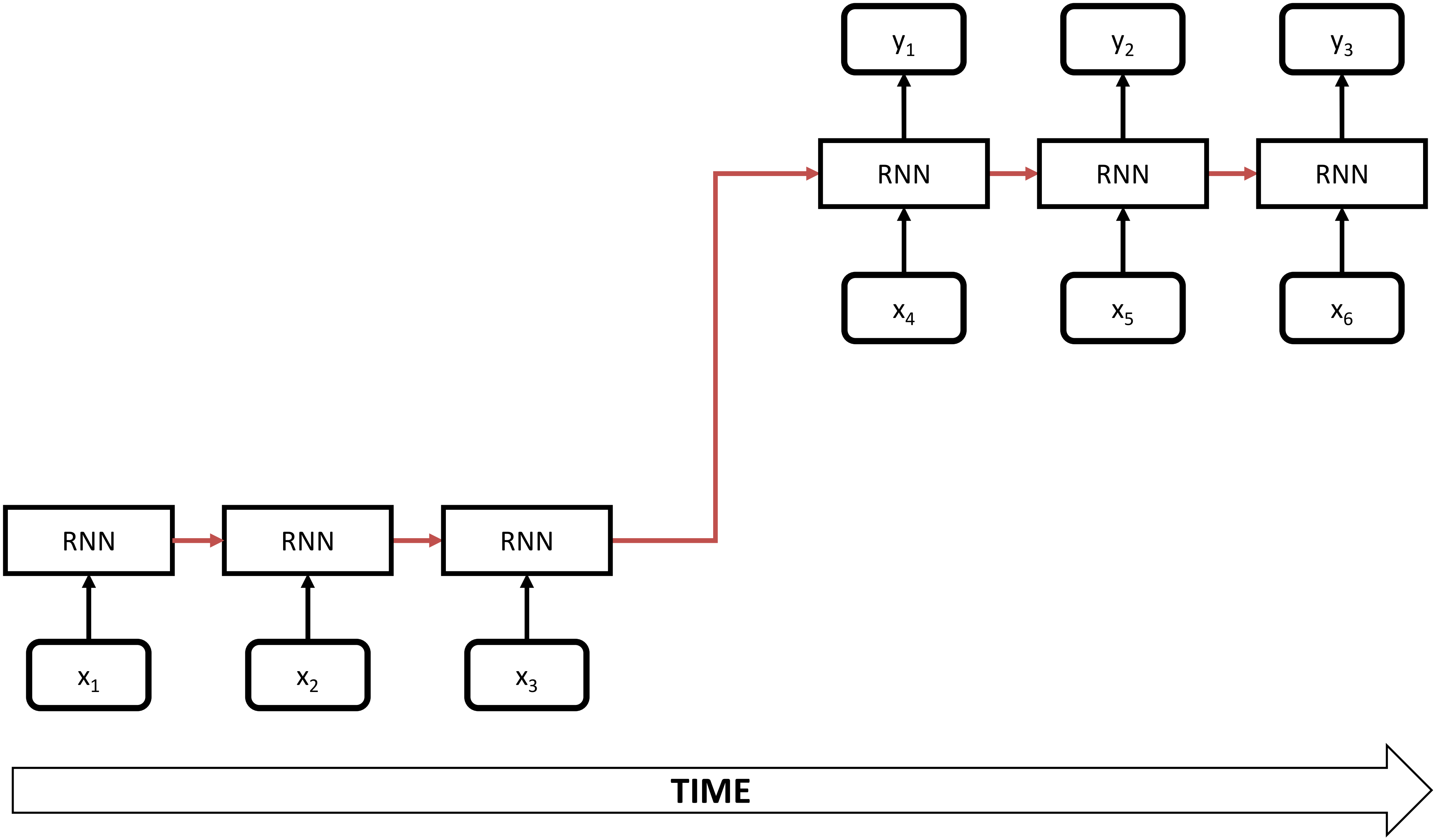
Many-to-Many RNN Network



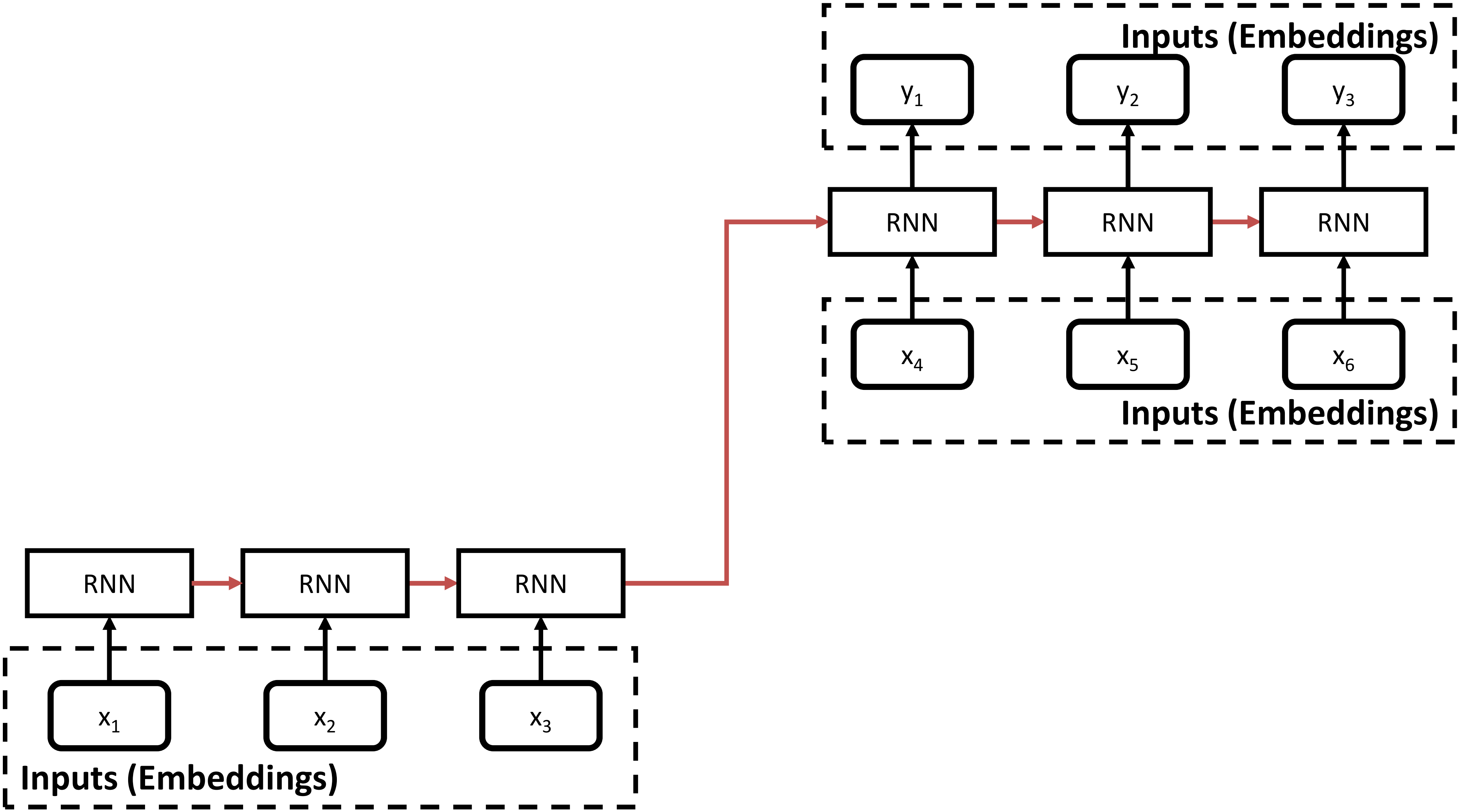
Sequence-to-Sequence (seq2seq)



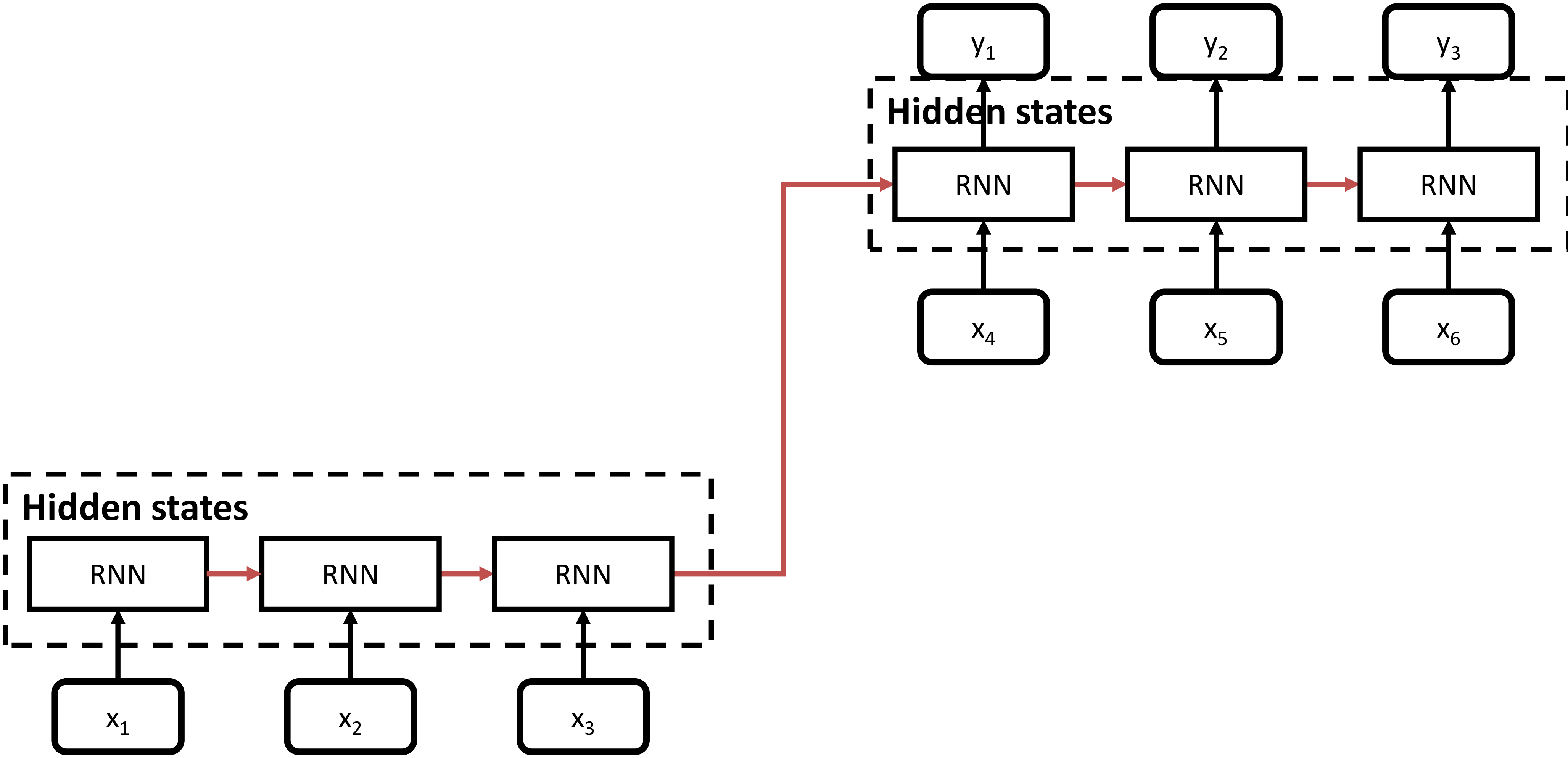
Sequence-to-Sequence (seq2seq)



Sequence-to-Sequence (seq2seq)

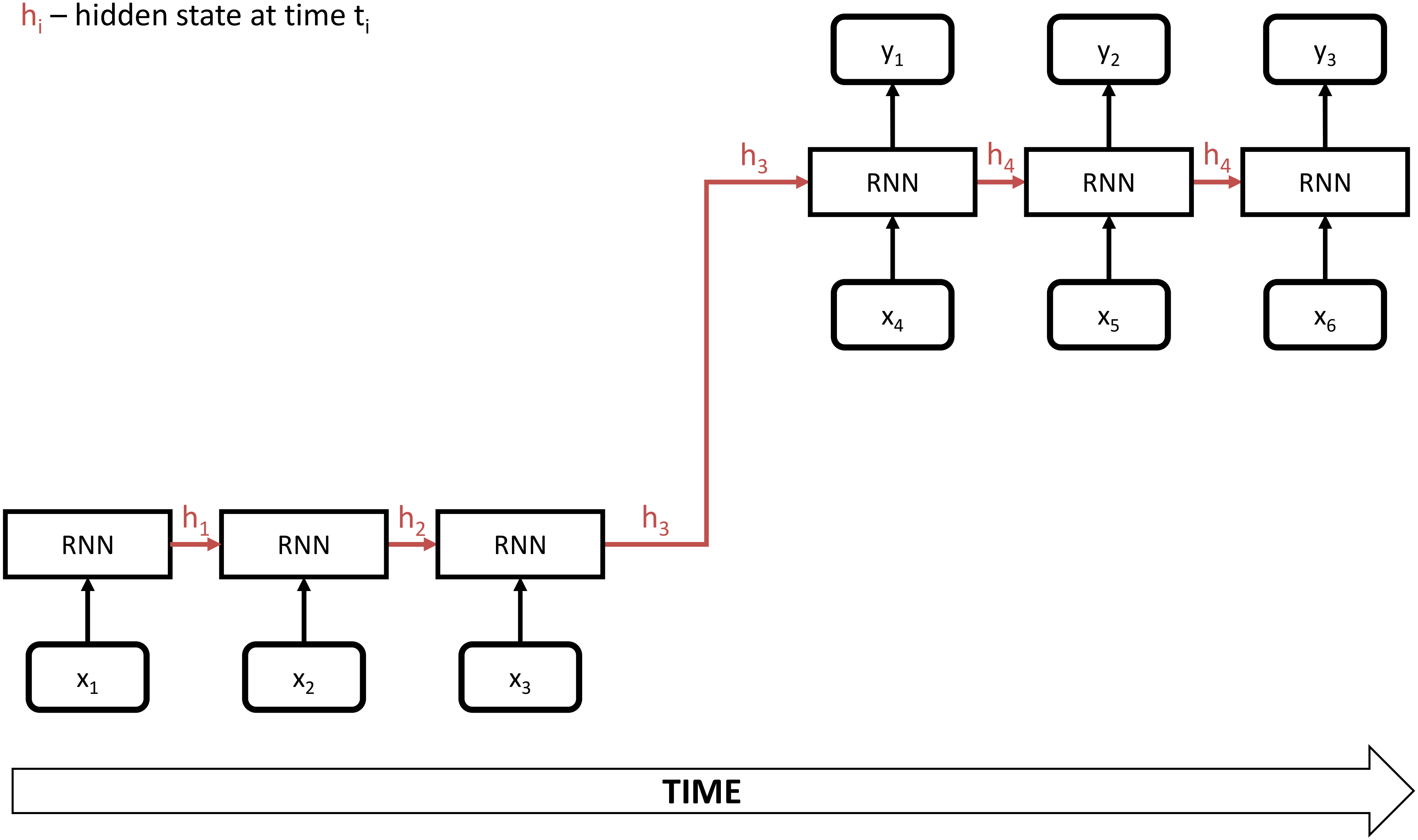


Sequence-to-Sequence (seq2seq)



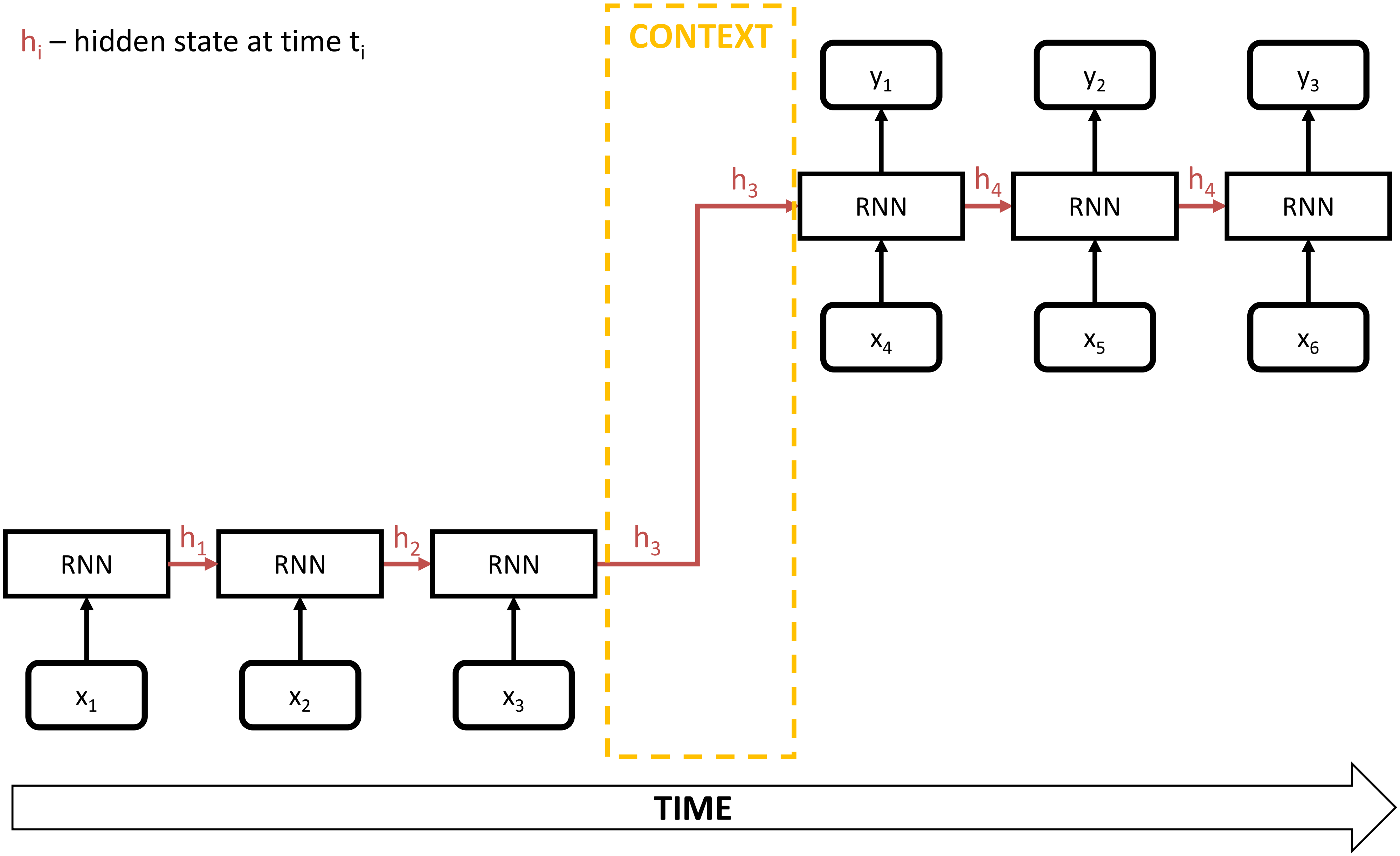
RNN Encoder-Decoder Architecture

h_i – hidden state at time t_i

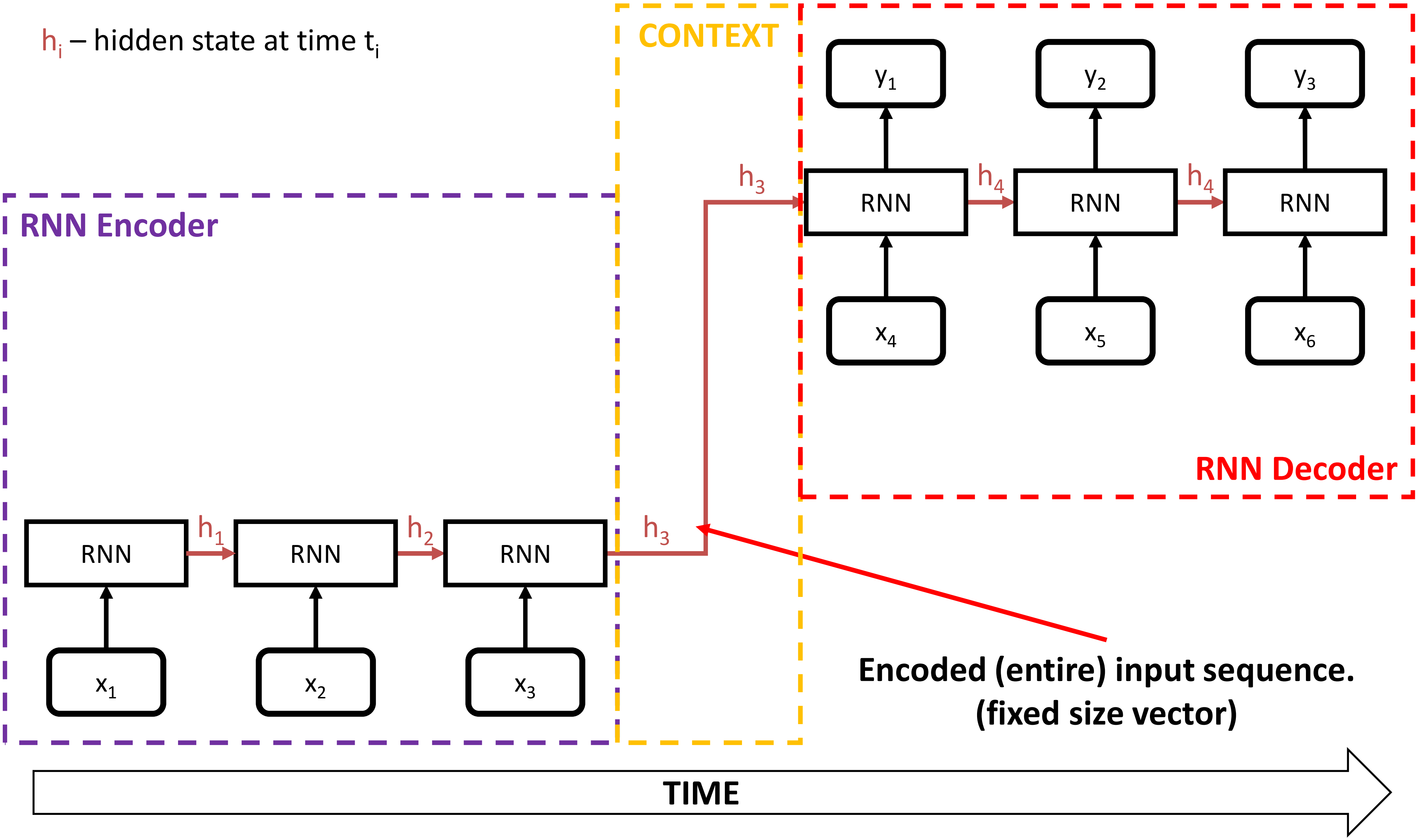


RNN Encoder-Decoder Architecture

h_i – hidden state at time t_i

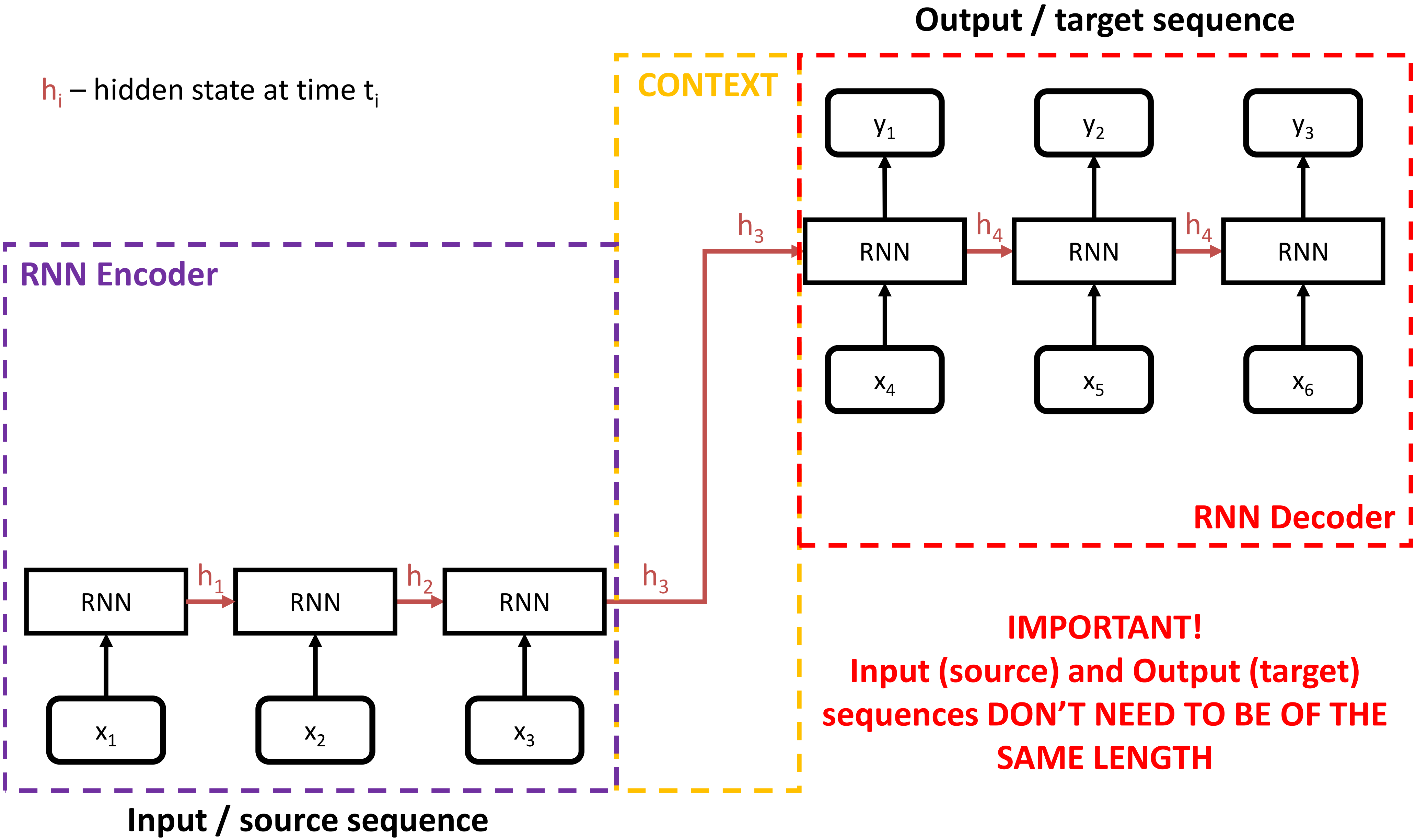


RNN Encoder-Decoder Architecture



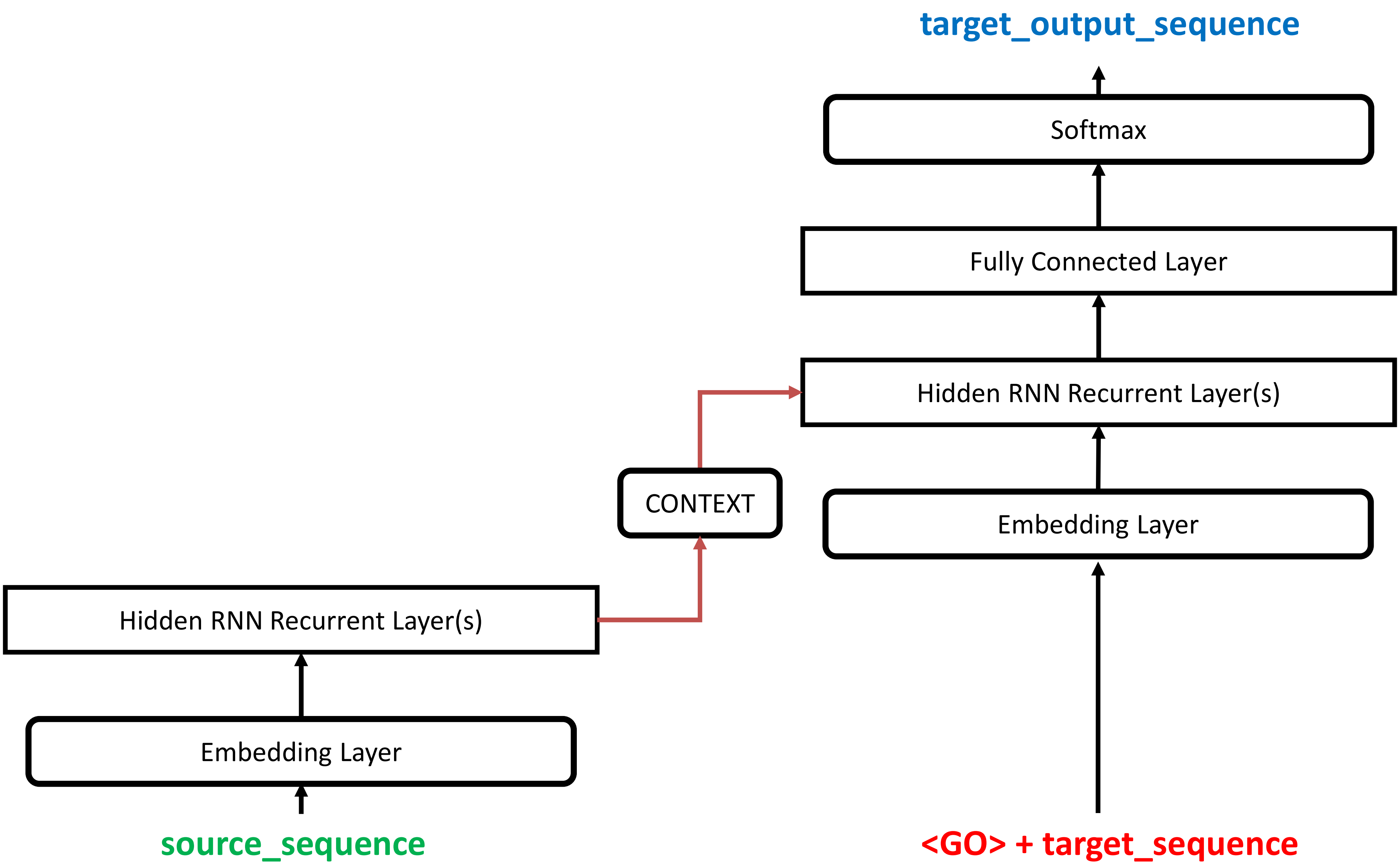
RNN Encoder-Decoder

h_i – hidden state at time t_i

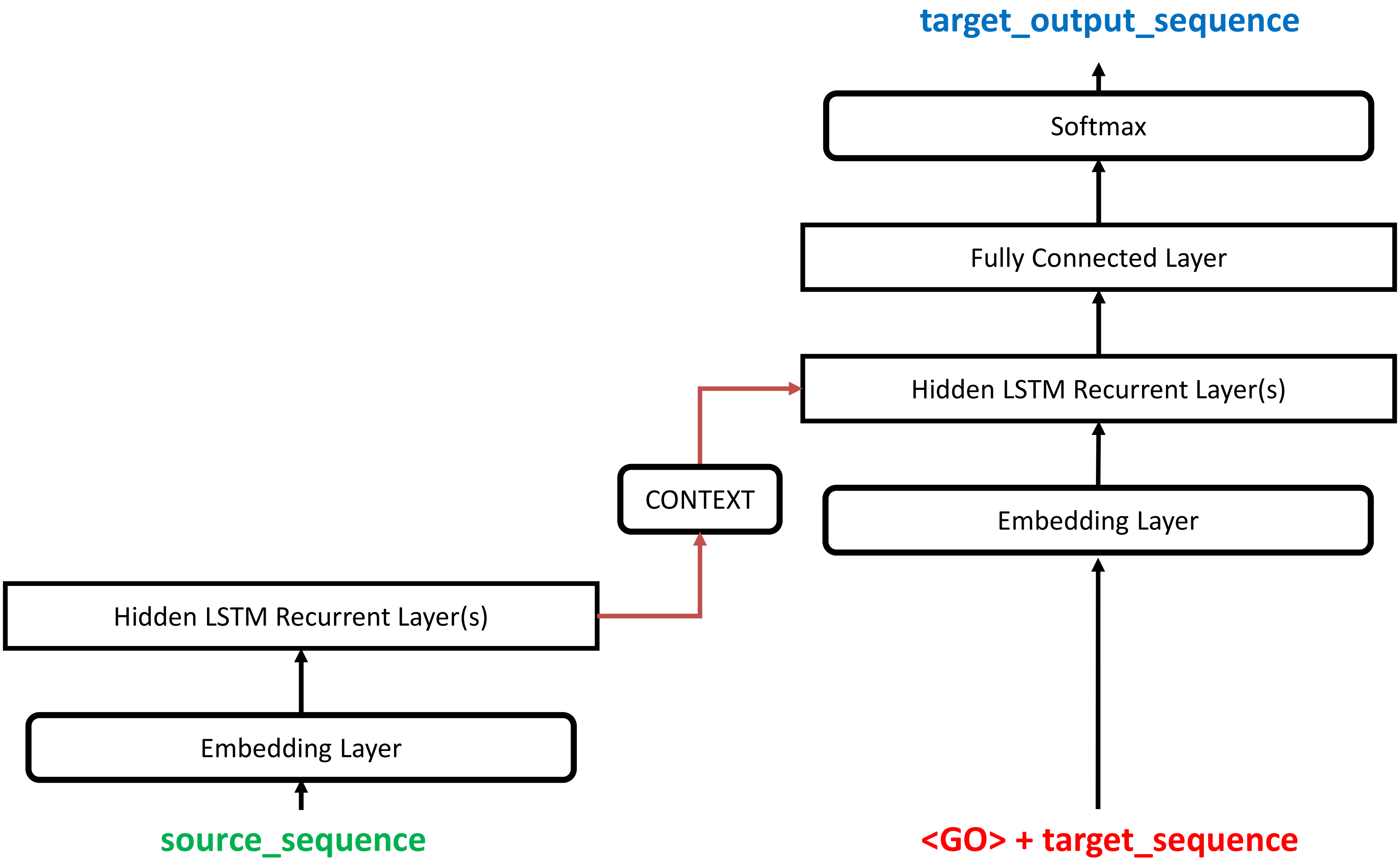


IMPORTANT!
Input (source) and Output (target) sequences DON'T NEED TO BE OF THE SAME LENGTH

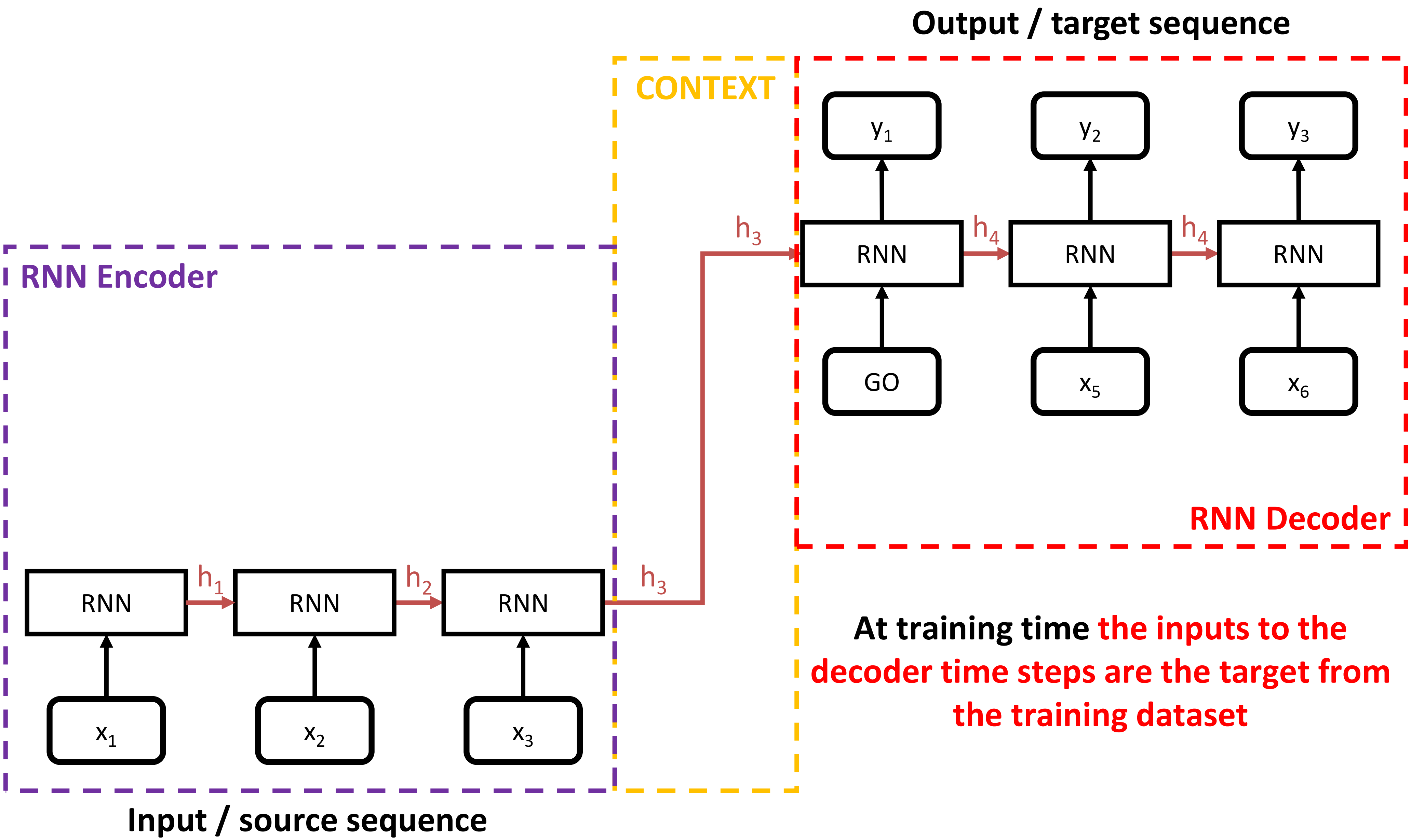
RNN Encoder-Decoder Architecture



RNN Encoder Decoder Architecture

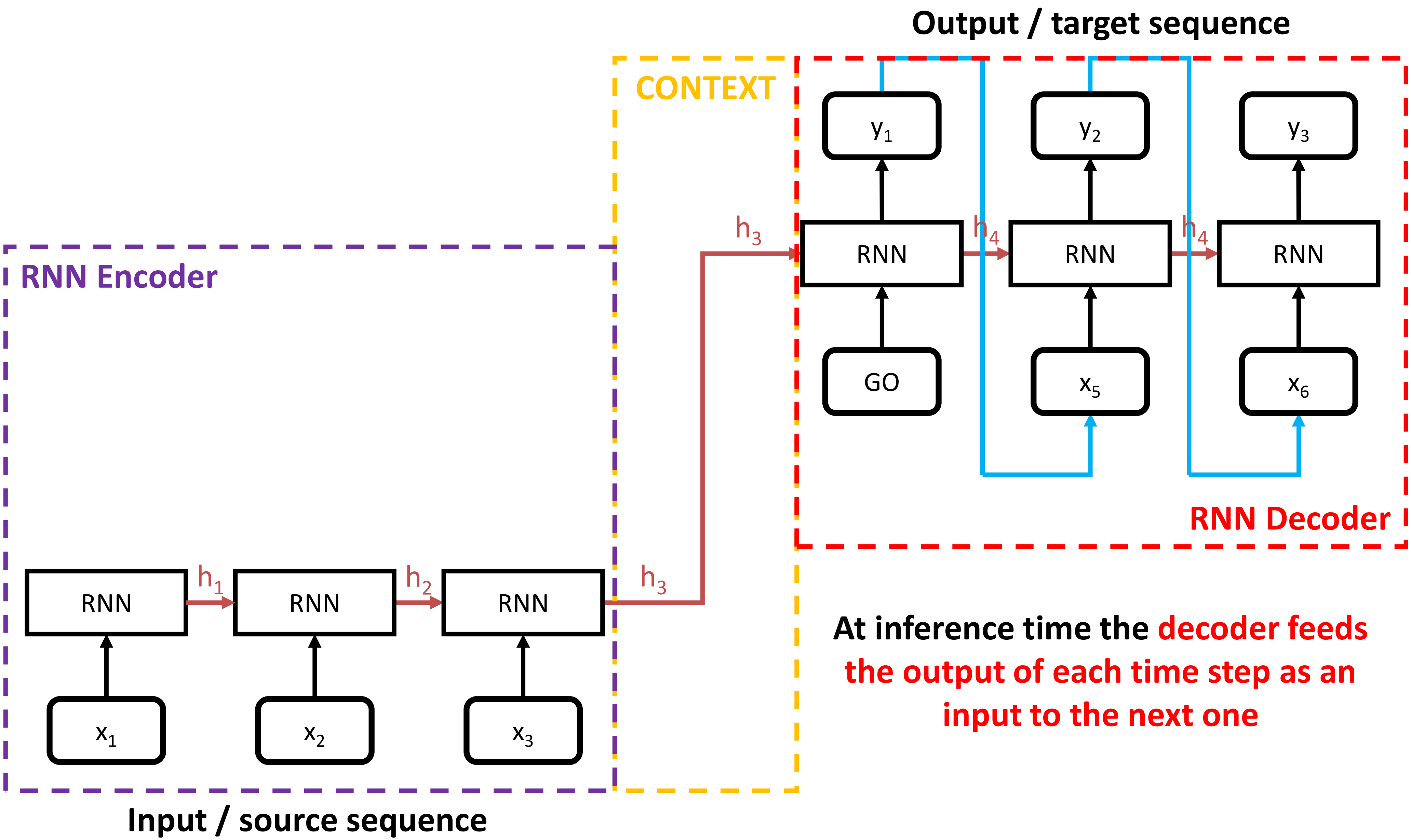


RNN Encoder-Decoder: Training



At training time the inputs to the decoder time steps are the target from the training dataset

RNN Encoder-Decoder: Inference

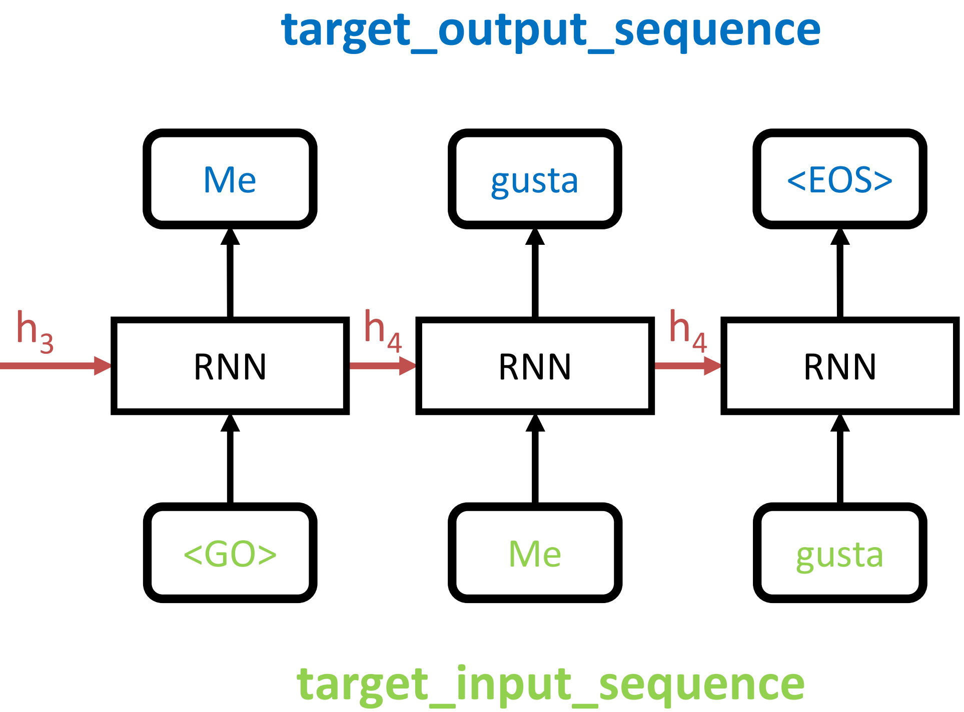


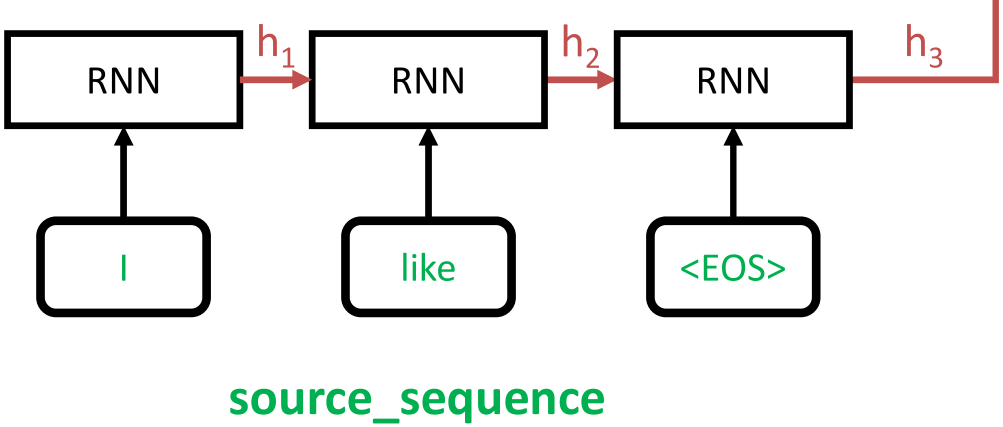
At inference time the decoder feeds the output of each time step as an input to the next one

RNN Encoder-Decoder: Data Prep

Data: <source_sequence, target_sequence> pairs

- Append <EOS> to the source_sequence
- Prepend <GO> (or <SOS>) to the target_sequence to obtain the target_input_sequence and append <EOS> to obtain target_output_sequence.
- Pad up to the max_input_length (max_target_length) within the batch using the <PAD> token.
- Encode tokens based of vocabulary (or embedding)
- Replace out of vocabulary (OOV) tokens with <UNK>. Compute the length of each input and target sequence in the batch.

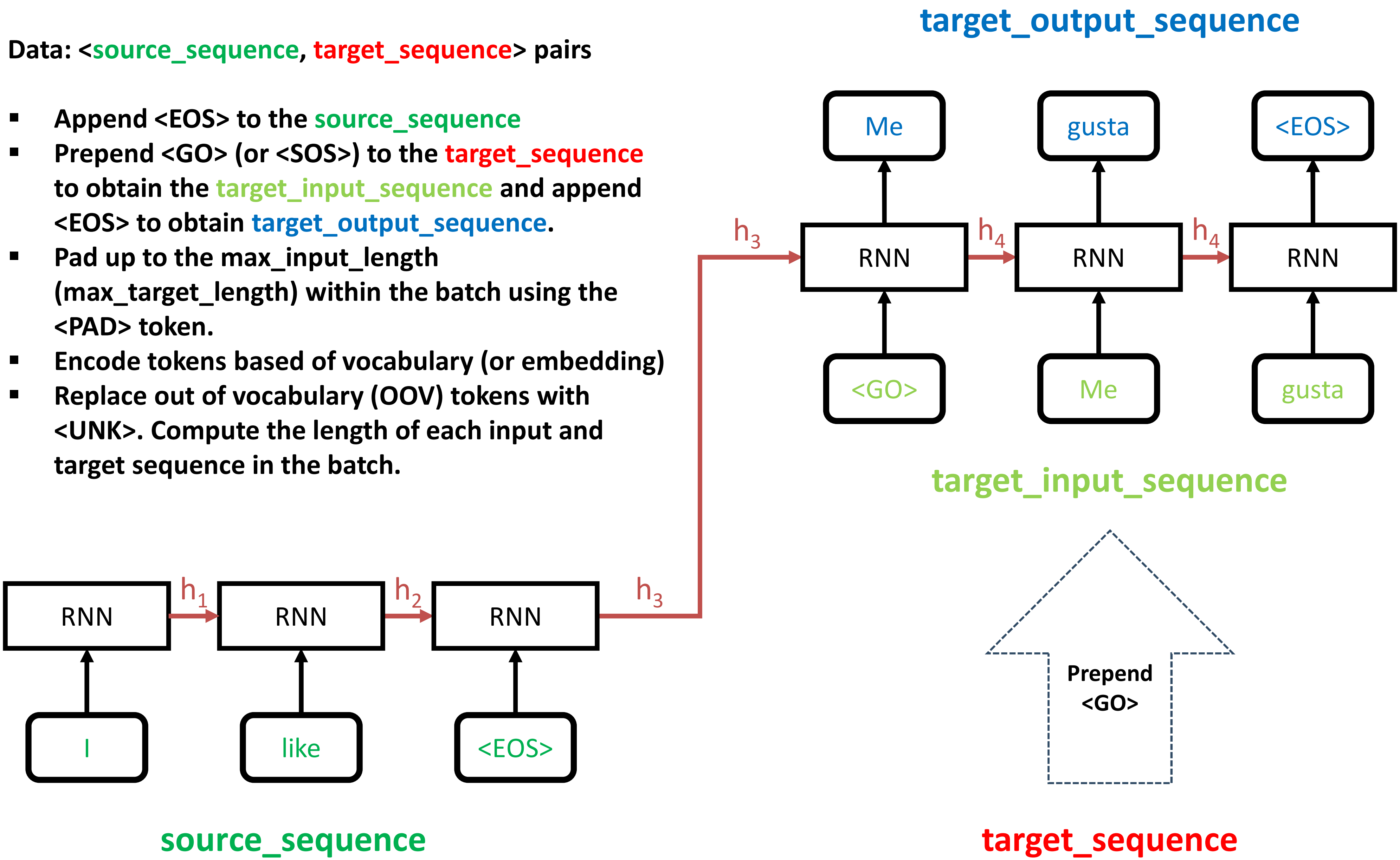




RNN Encoder-Decoder: Training

Data: <source_sequence, target_sequence> pairs

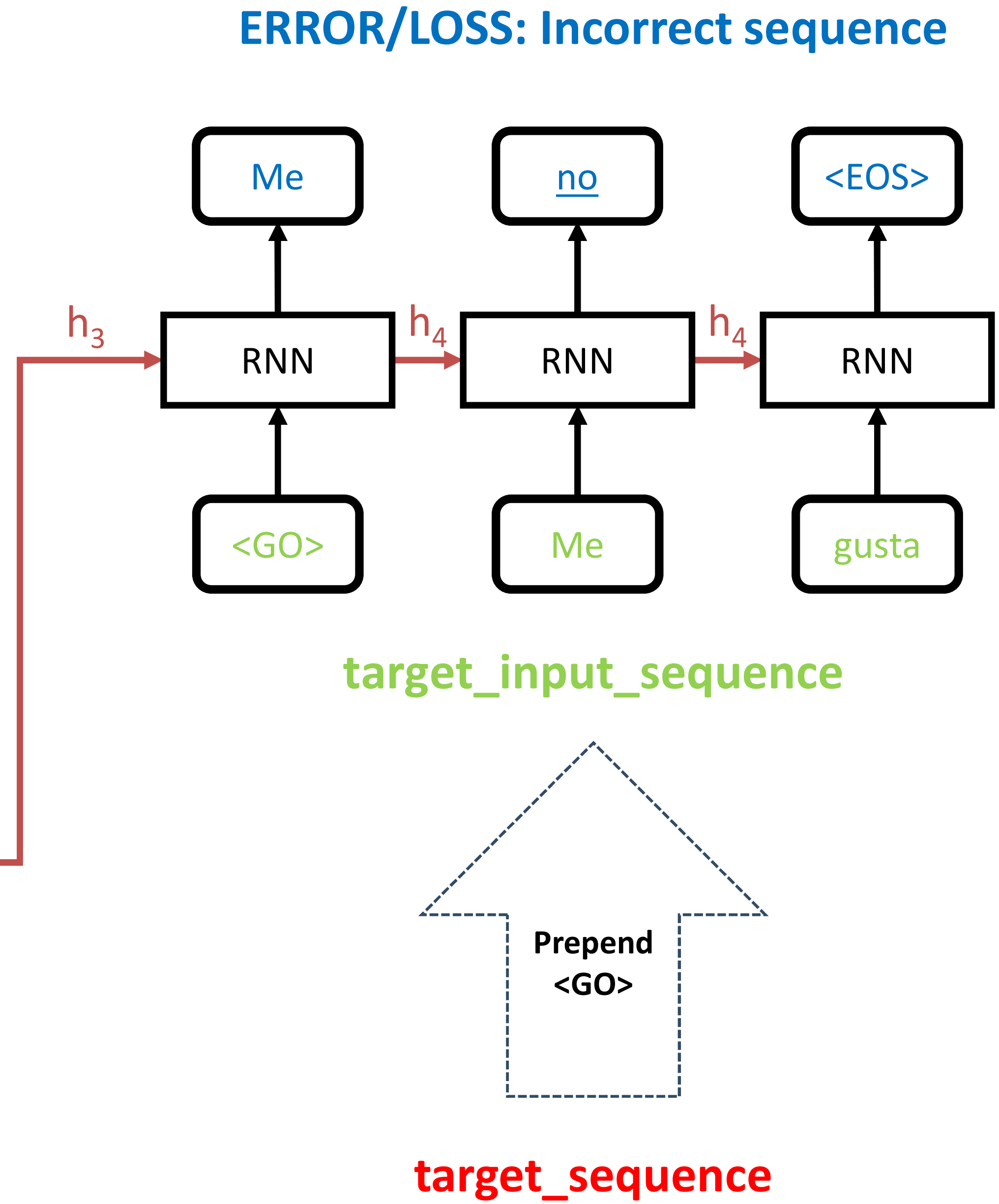
- **Append <EOS> to the source_sequence**
- **Prepend <GO> (or <SOS>) to the target_sequence to obtain the target_input_sequence and append <EOS> to obtain target_output_sequence.**
- **Pad up to the max_input_length (max_target_length) within the batch using the <PAD> token.**
- **Encode tokens based of vocabulary (or embedding)**
- **Replace out of vocabulary (OOV) tokens with <UNK>. Compute the length of each input and target sequence in the batch.**

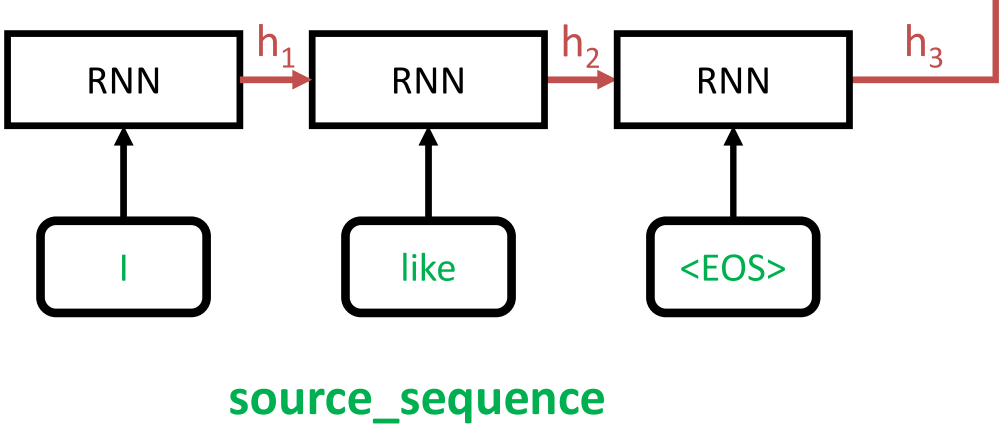


RNN Encoder-Decoder: Training

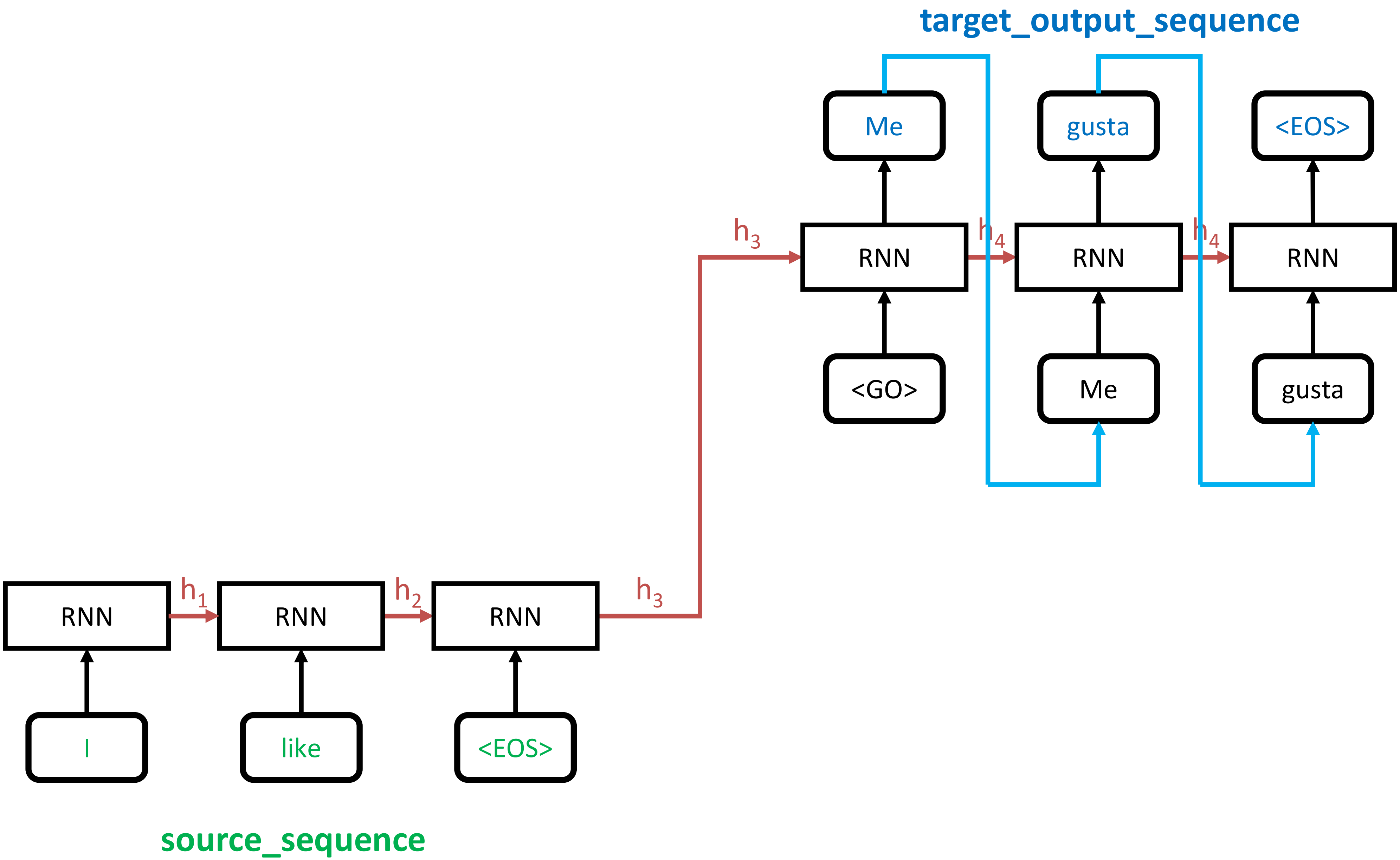
Data: <source_sequence, target_sequence> pairs

- **Append <EOS> to the source_sequence**
- **Prepend <GO> (or <SOS>) to the target_sequence to obtain the target_input_sequence and append <EOS> to obtain target_output_sequence.**
- **Pad up to the max_input_length (max_target_length) within the batch using the <PAD> token.**
- **Encode tokens based of vocabulary (or embedding)**
- **Replace out of vocabulary (OOV) tokens with <UNK>. Compute the length of each input and target sequence in the batch.**

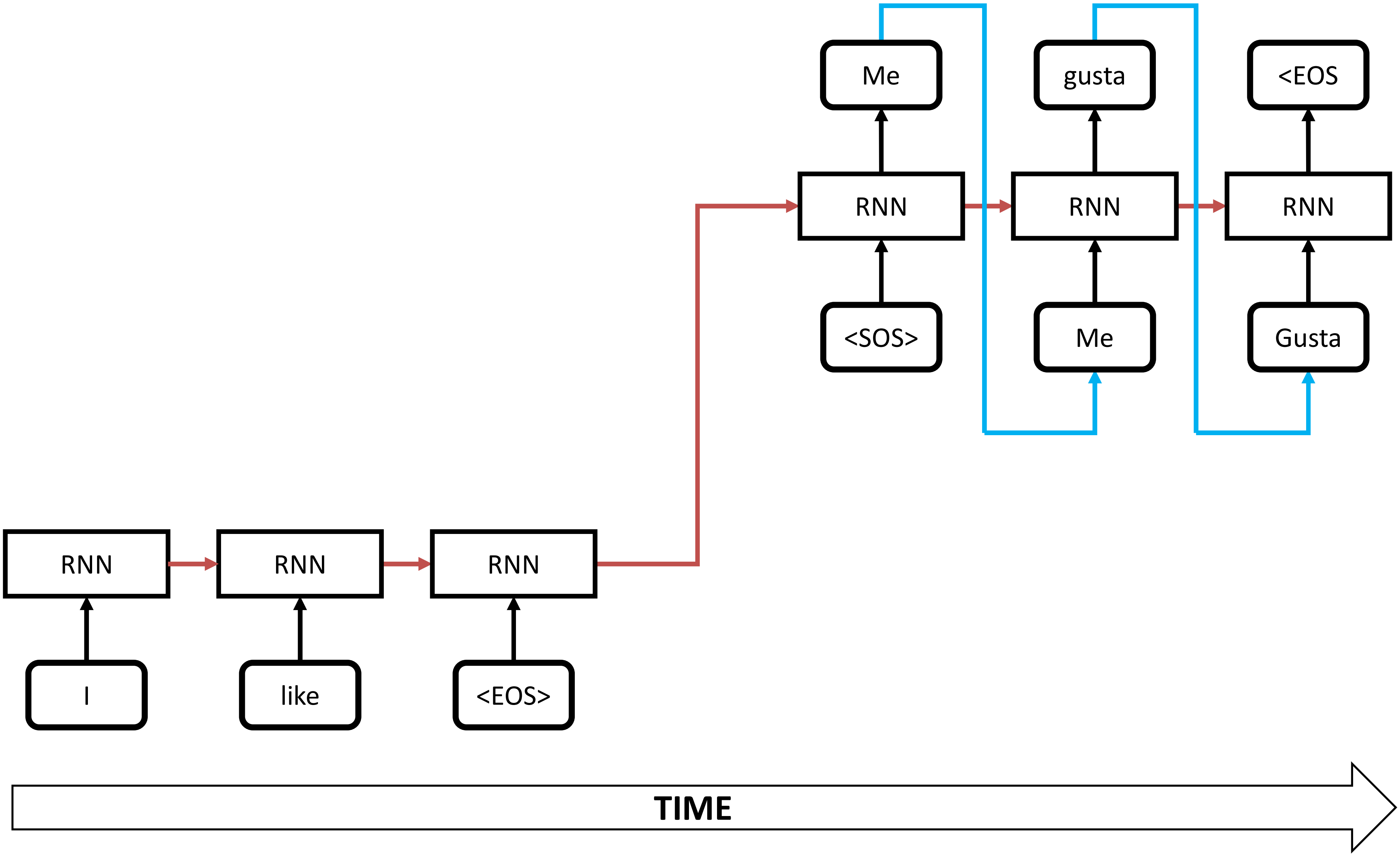




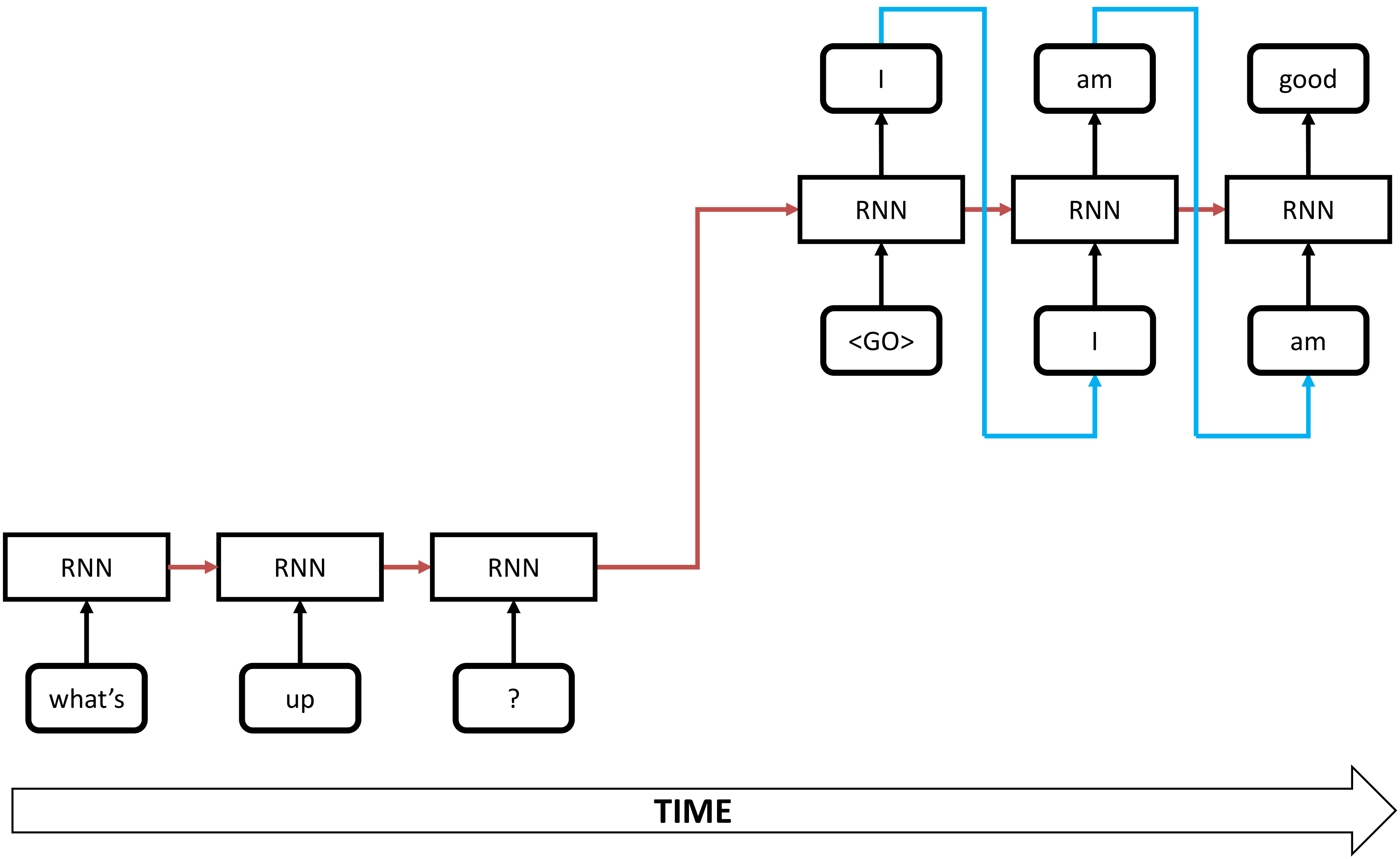
RNN Encoder-Decoder: Inference



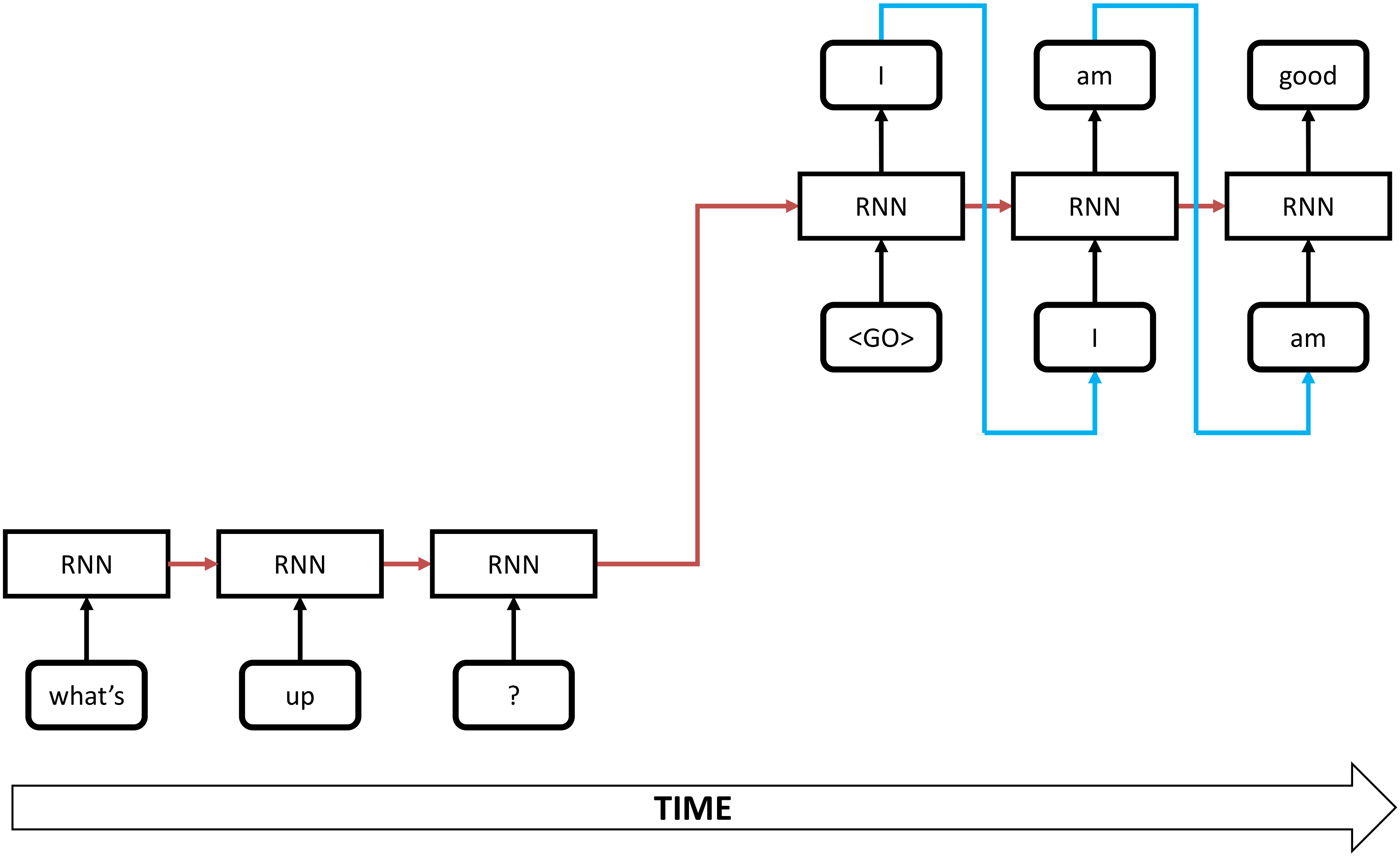
Encoder-Decoder: Translation



Encoder-Decoder:Question Answering

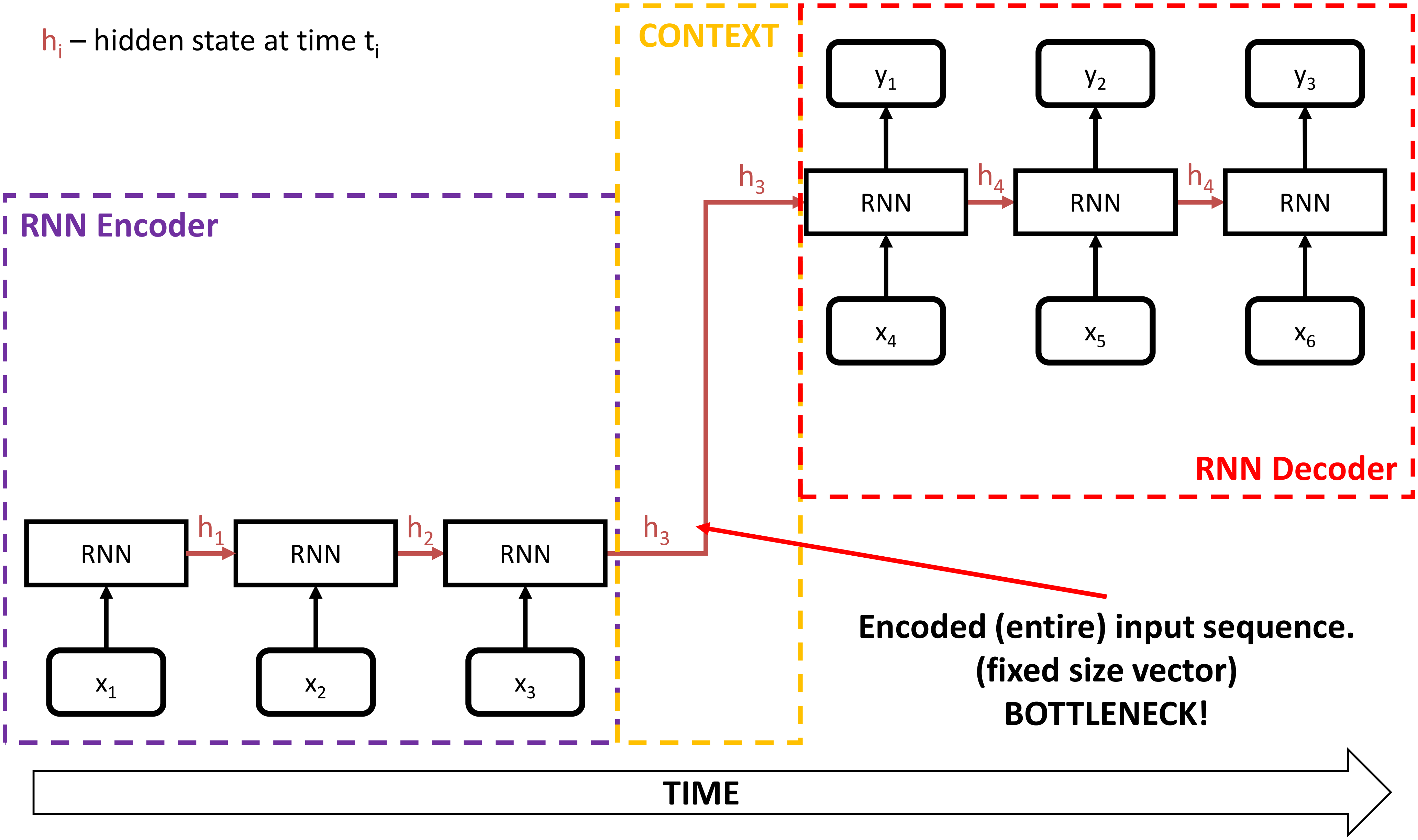


Encoder-Decoder: Chat Bot

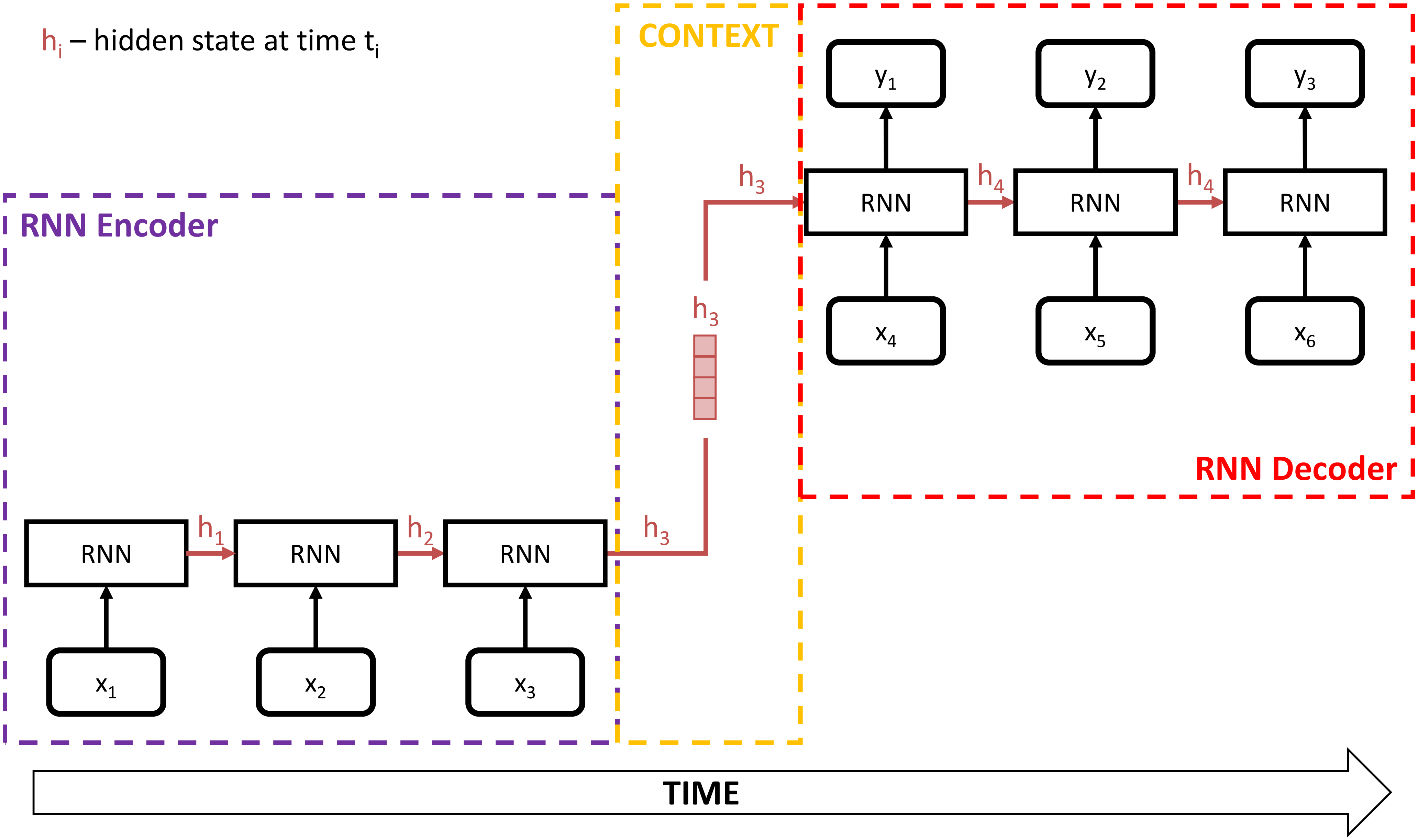


Sequence to Sequence Networks (seq2seq) With Attention

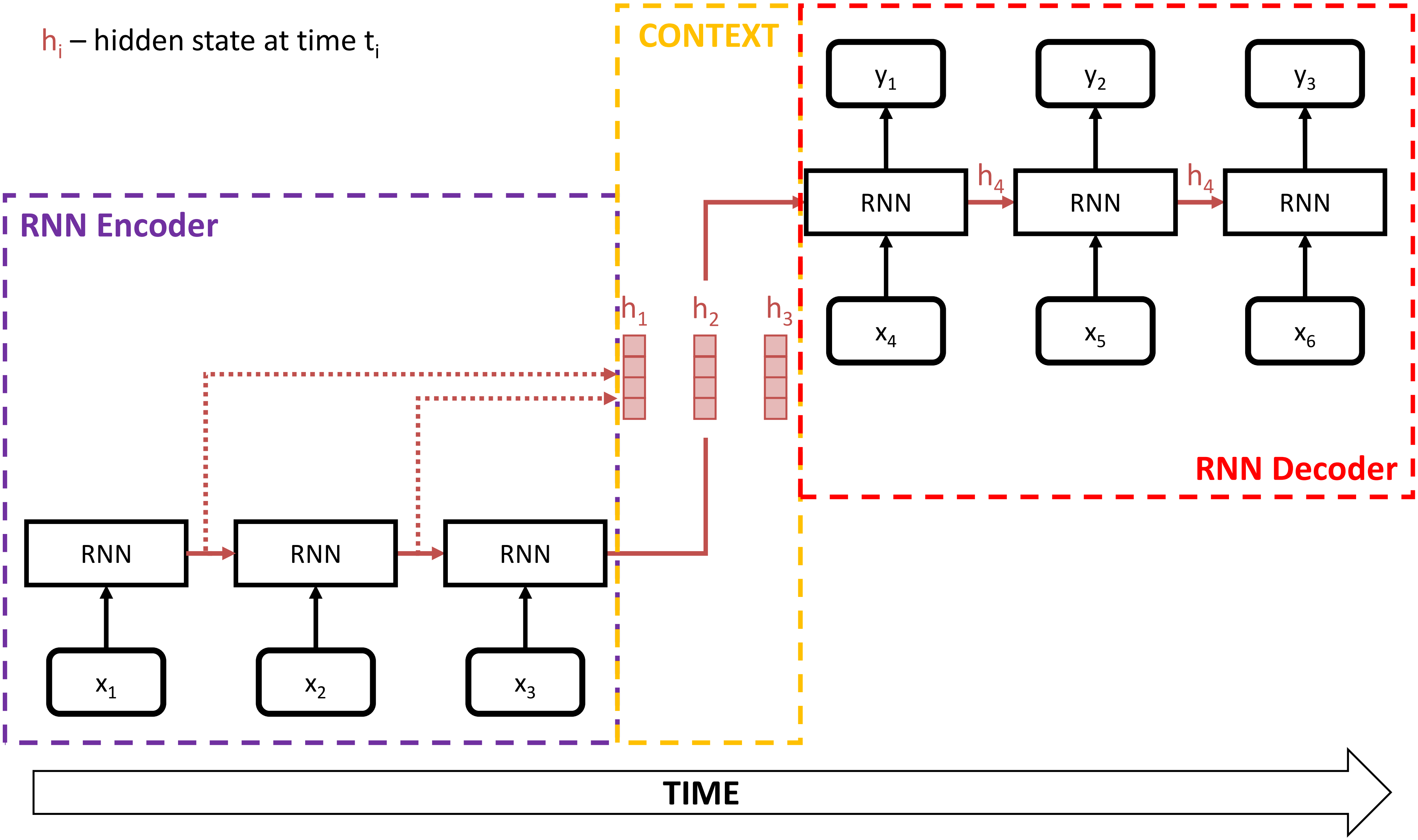
RNN Encoder-Decoder: Context



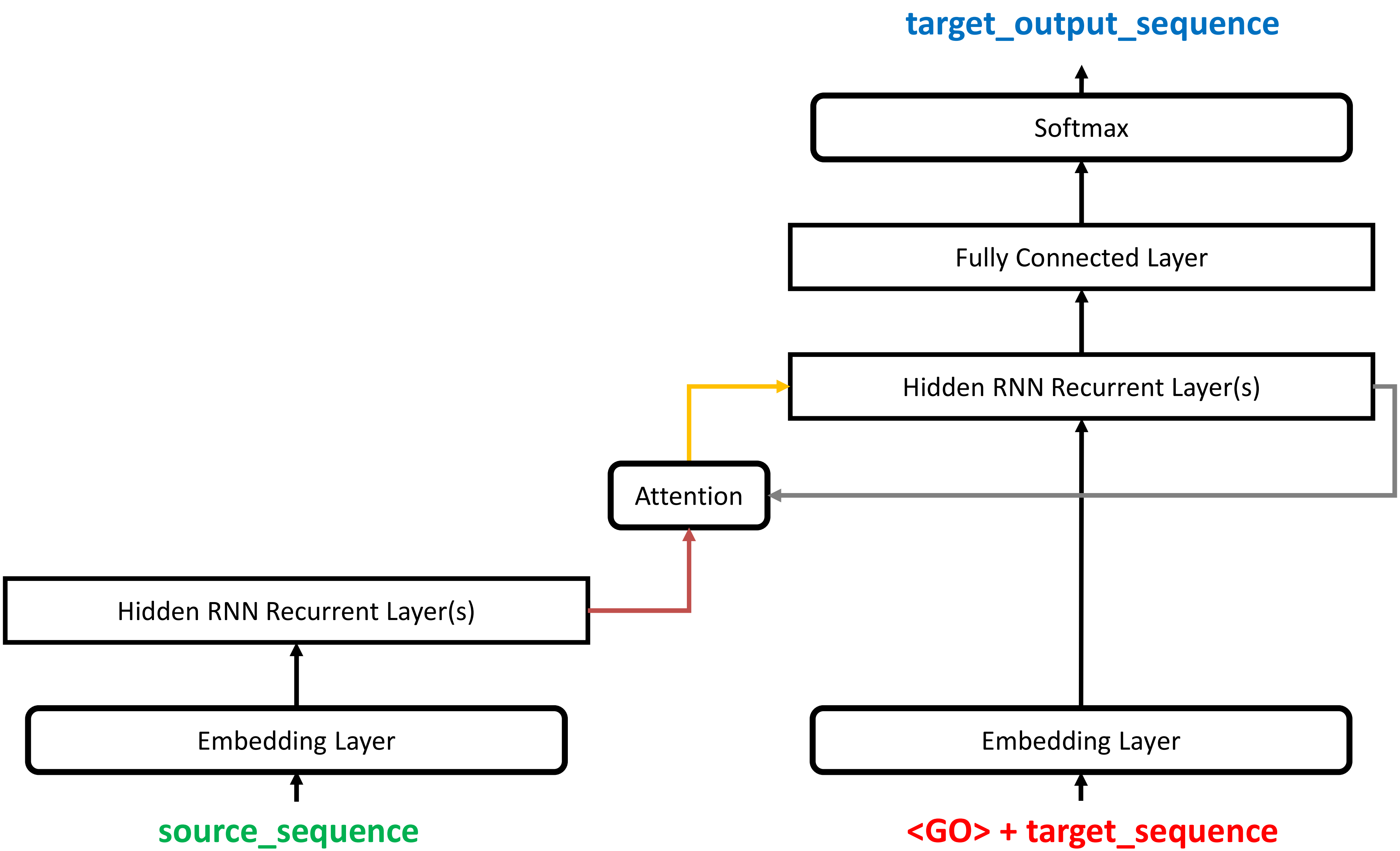
Fixed Length Context



RNN Encoder-Decoder Architecture



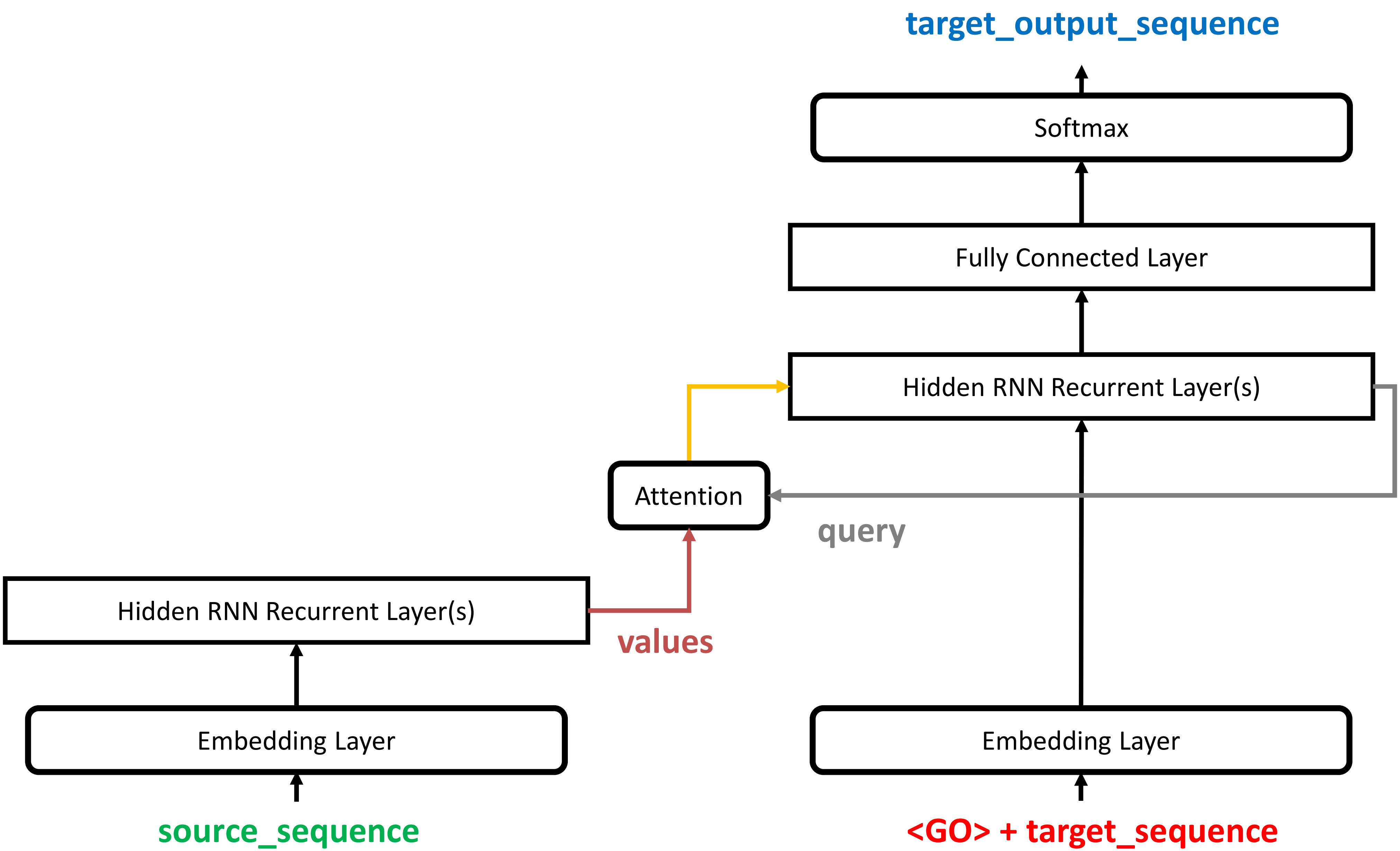
RNN Encoder-Decoder with Attention



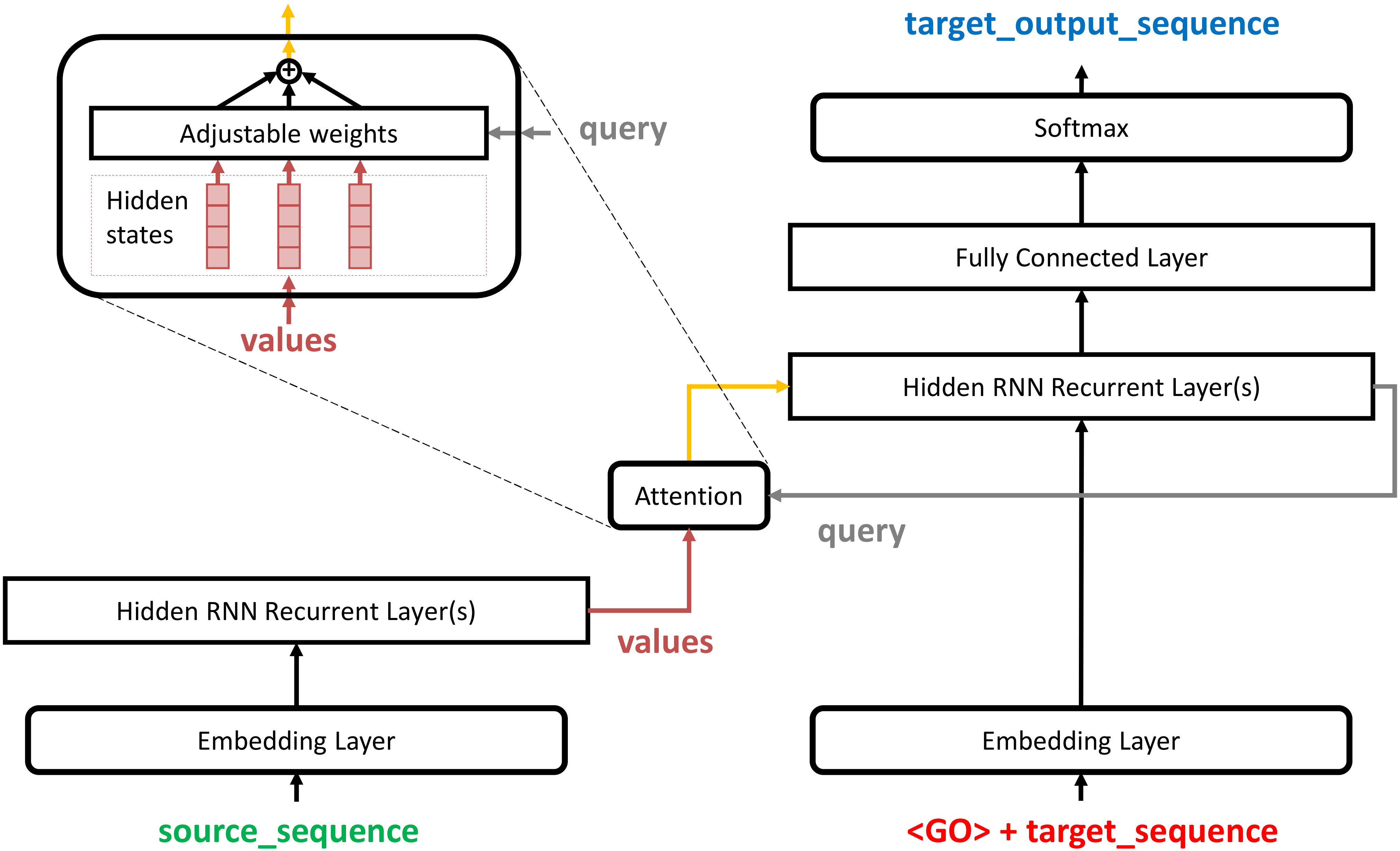
Attention Mechanism

- Given a set of vector values, and a vector query, attention is a technique to compute a weighted sum of the values, dependent on the query
- Attention mechanism "amplifies" important aspects of the signal from the encoder based on the decoder query
- In seq2seq models with attention, each decoder hidden state (query) attends to all the encoder hidden states (values)

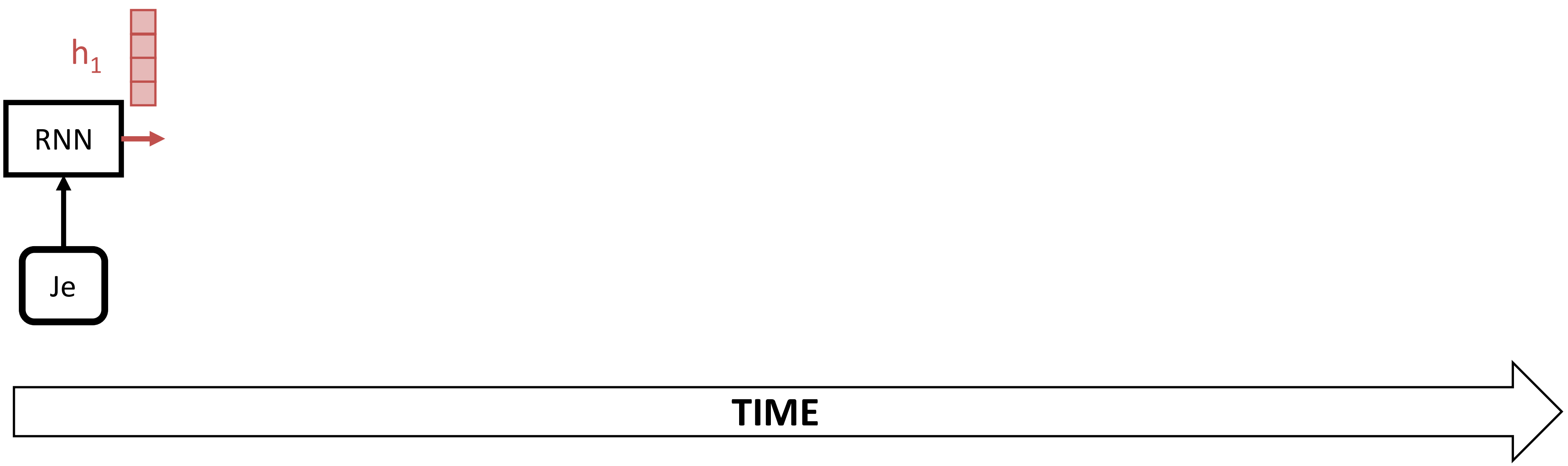
RNN Encoder-Decoder with Attention



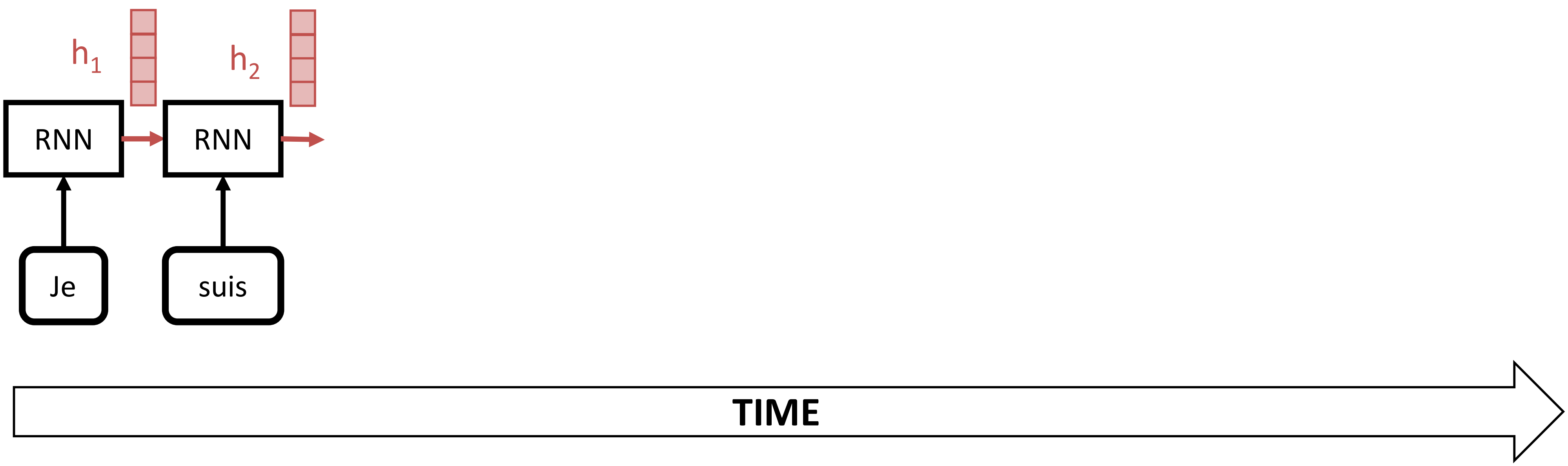
RNN Encoder-Decoder with Attention



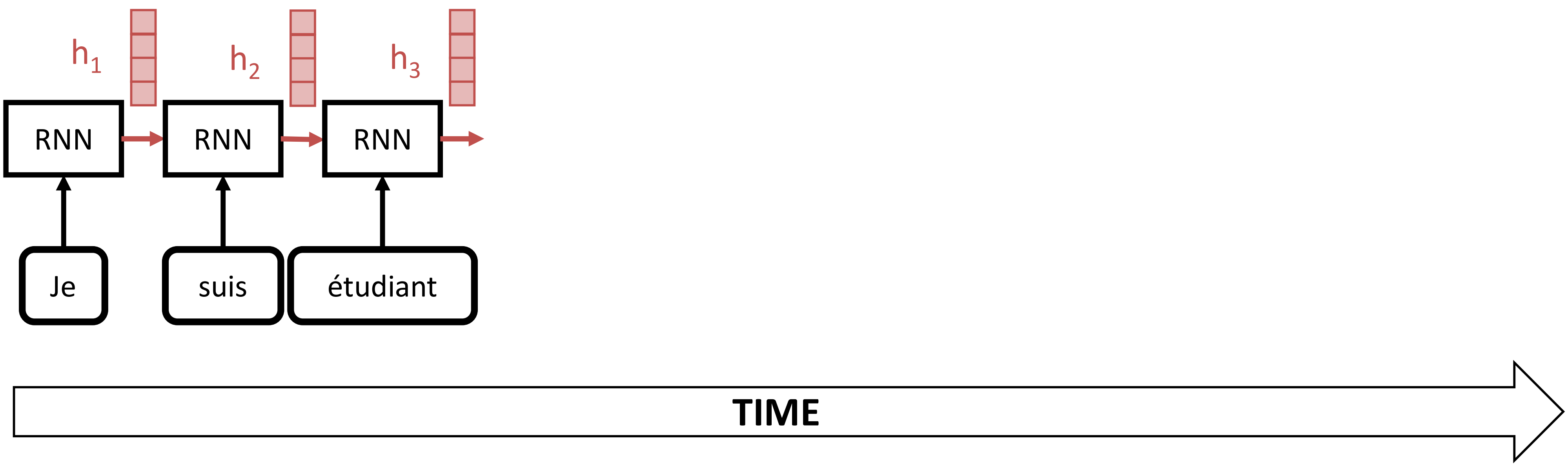
RNN Encoder-Decoder with Attention



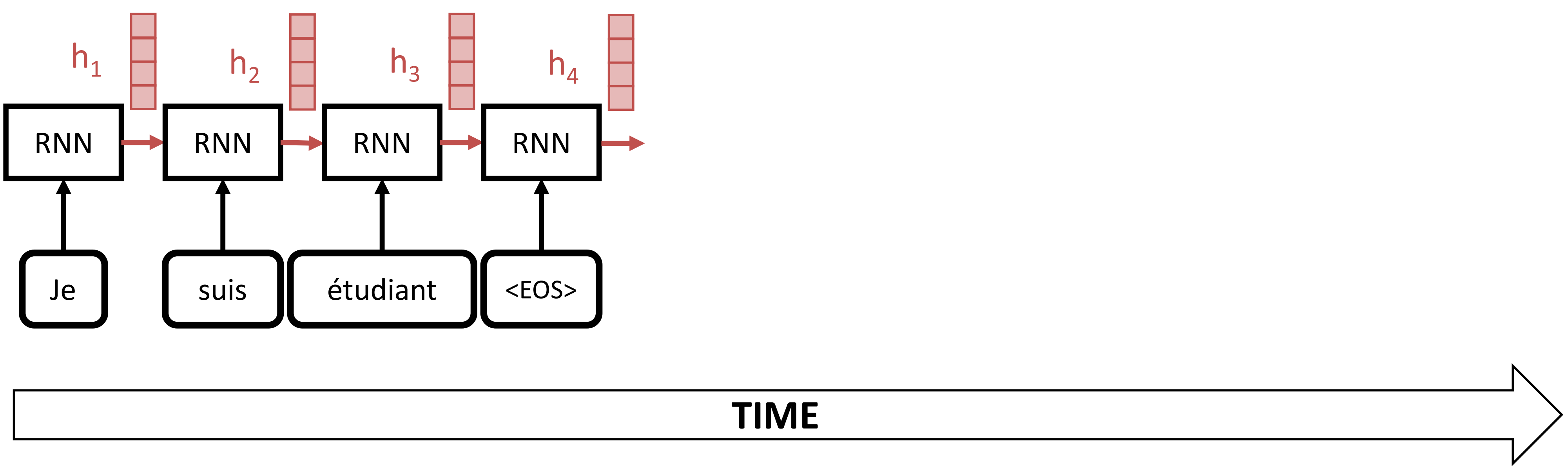
RNN Encoder-Decoder with Attention



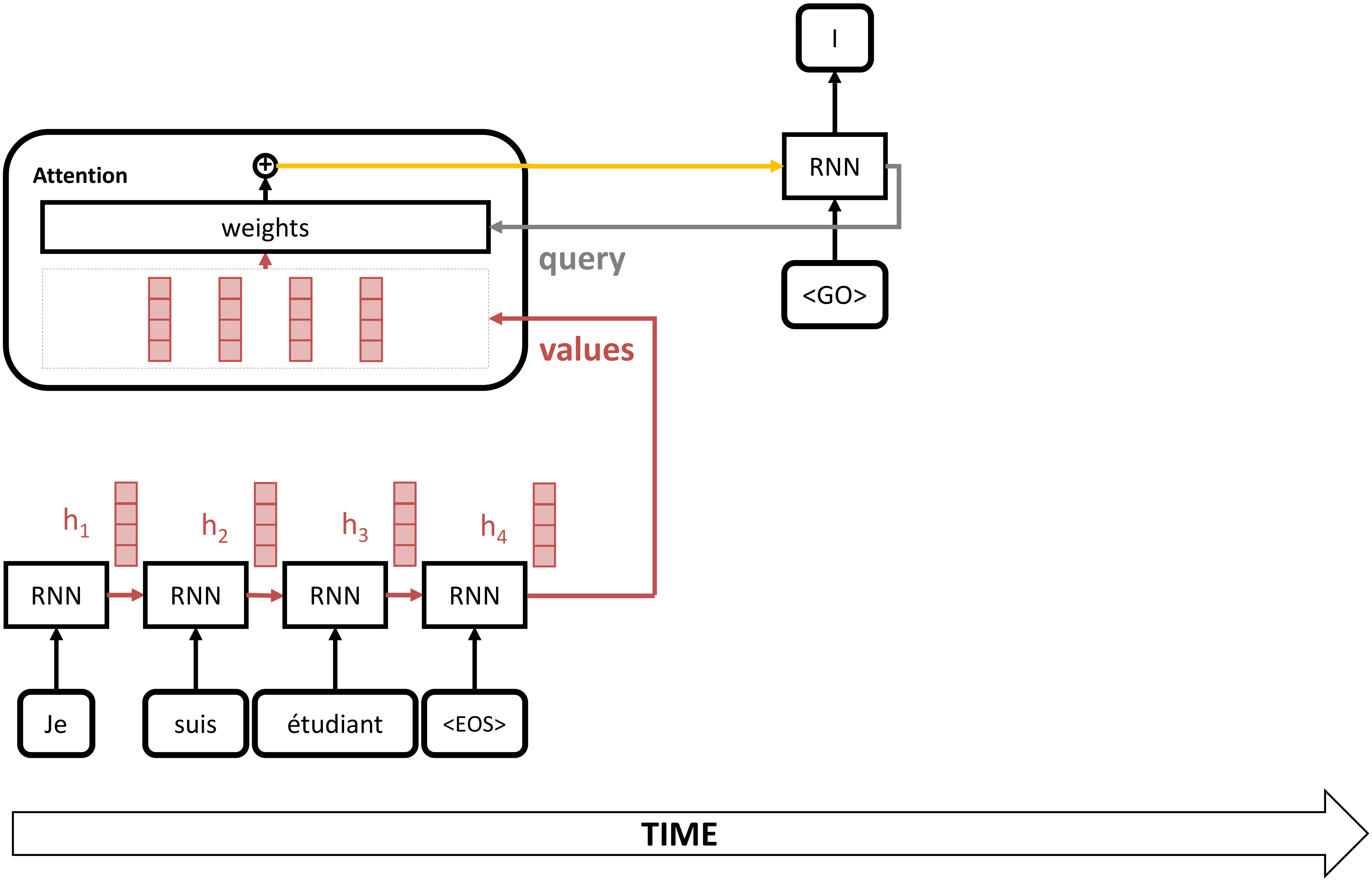
RNN Encoder-Decoder with Attention



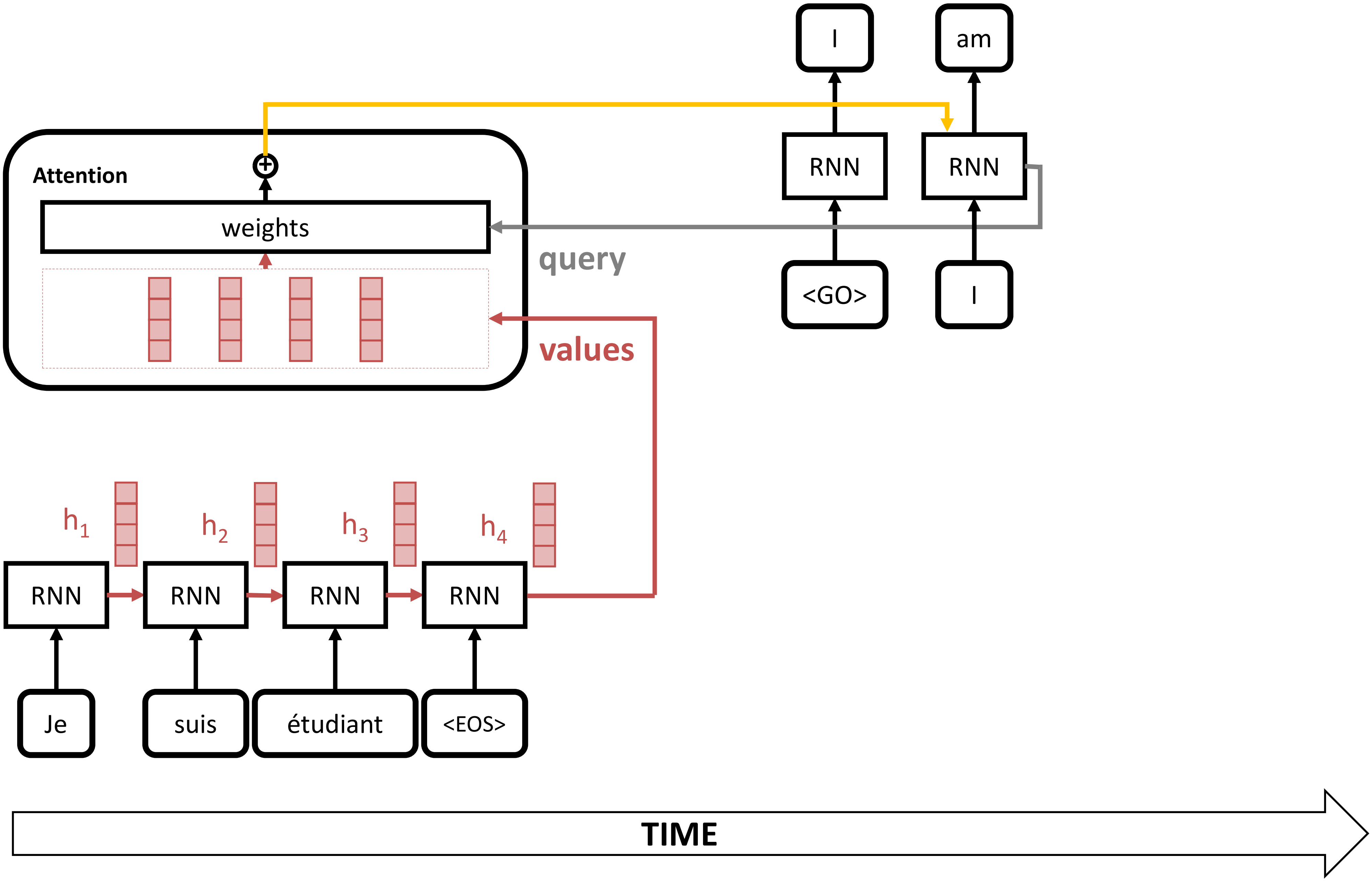
RNN Encoder-Decoder with Attention



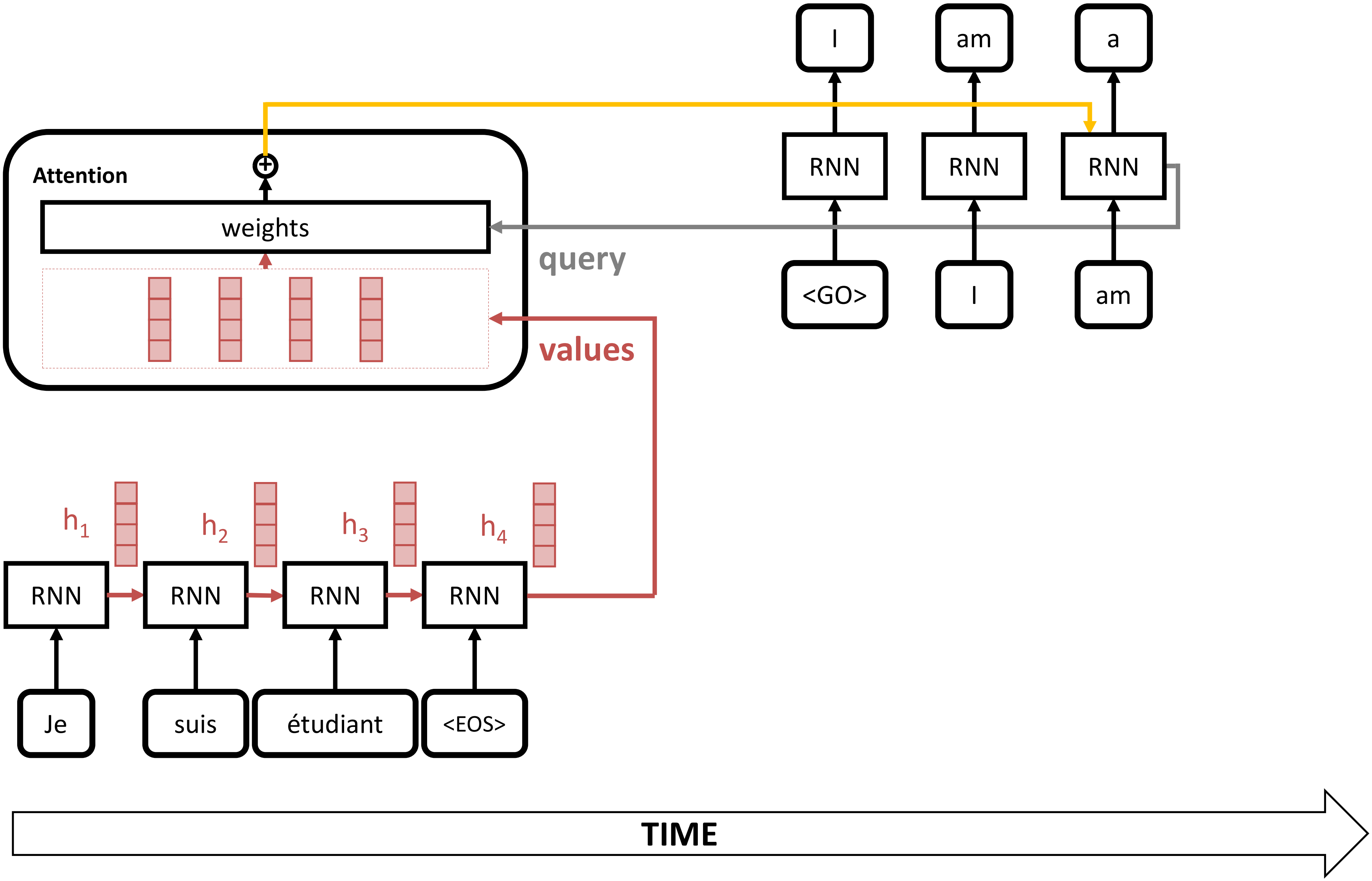
RNN Encoder-Decoder with Attention



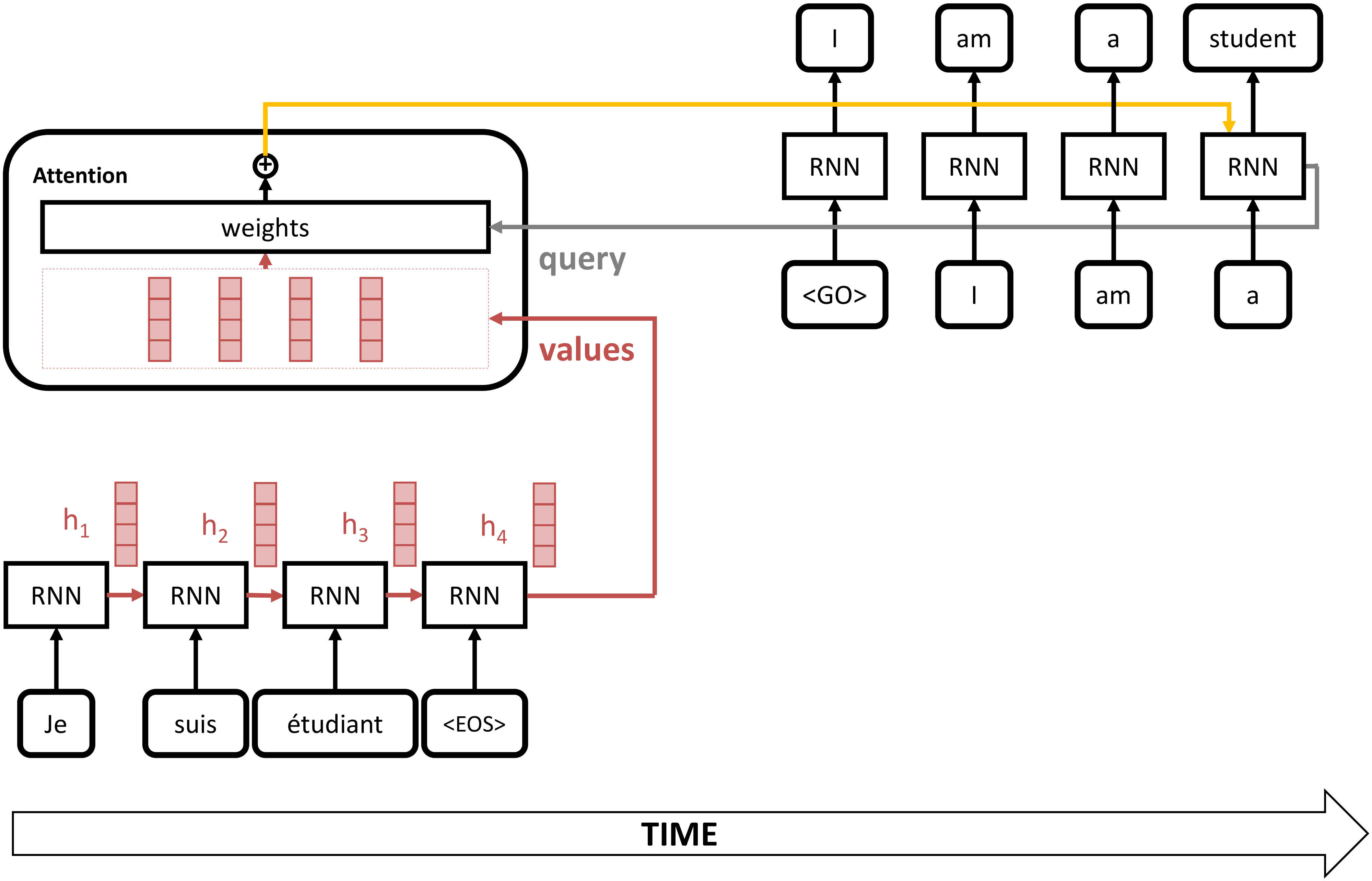
RNN Encoder-Decoder with Attention



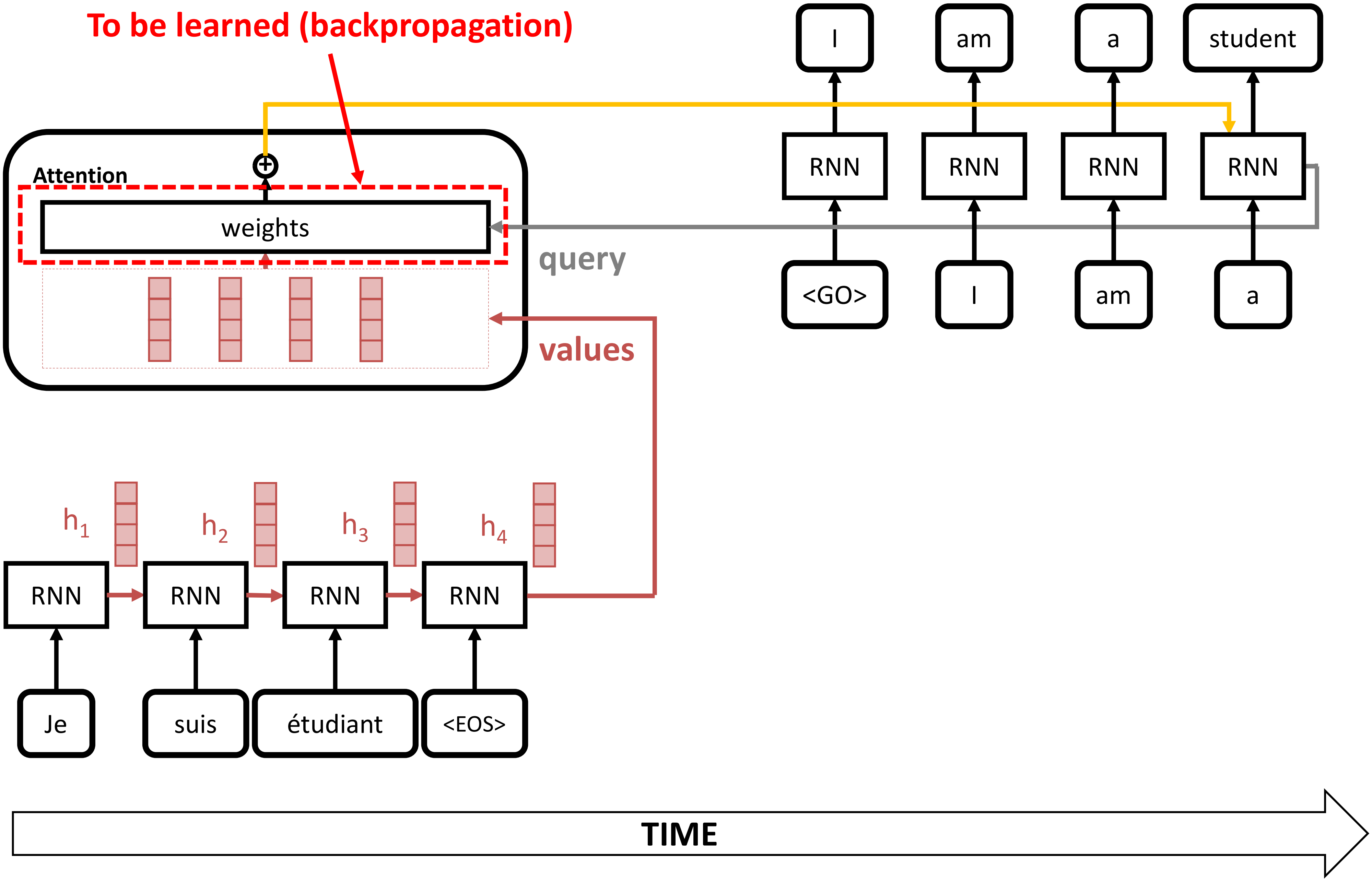
RNN Encoder-Decoder with Attention



RNN Encoder-Decoder with Attention



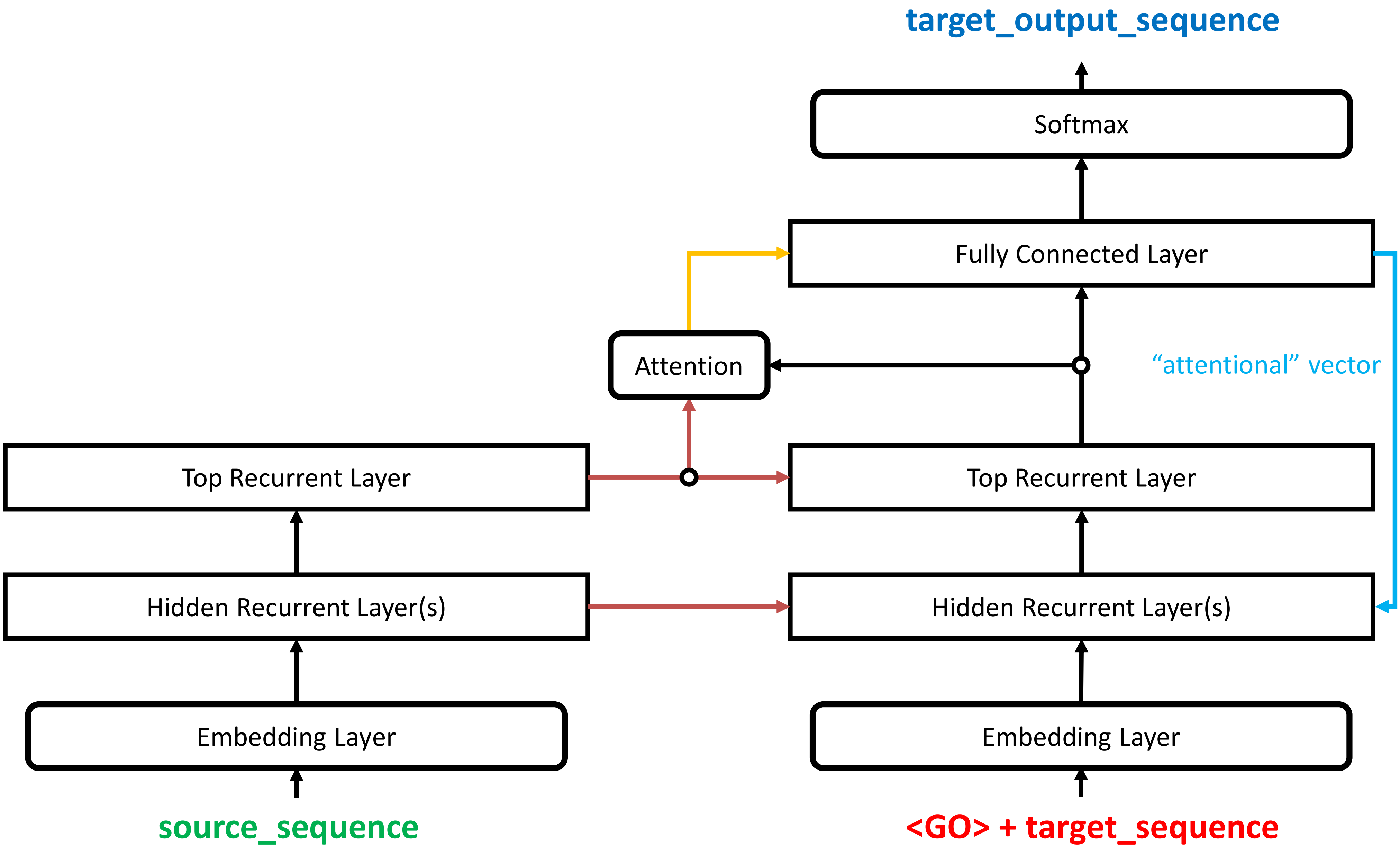
RNN Encoder-Decoder with Attention



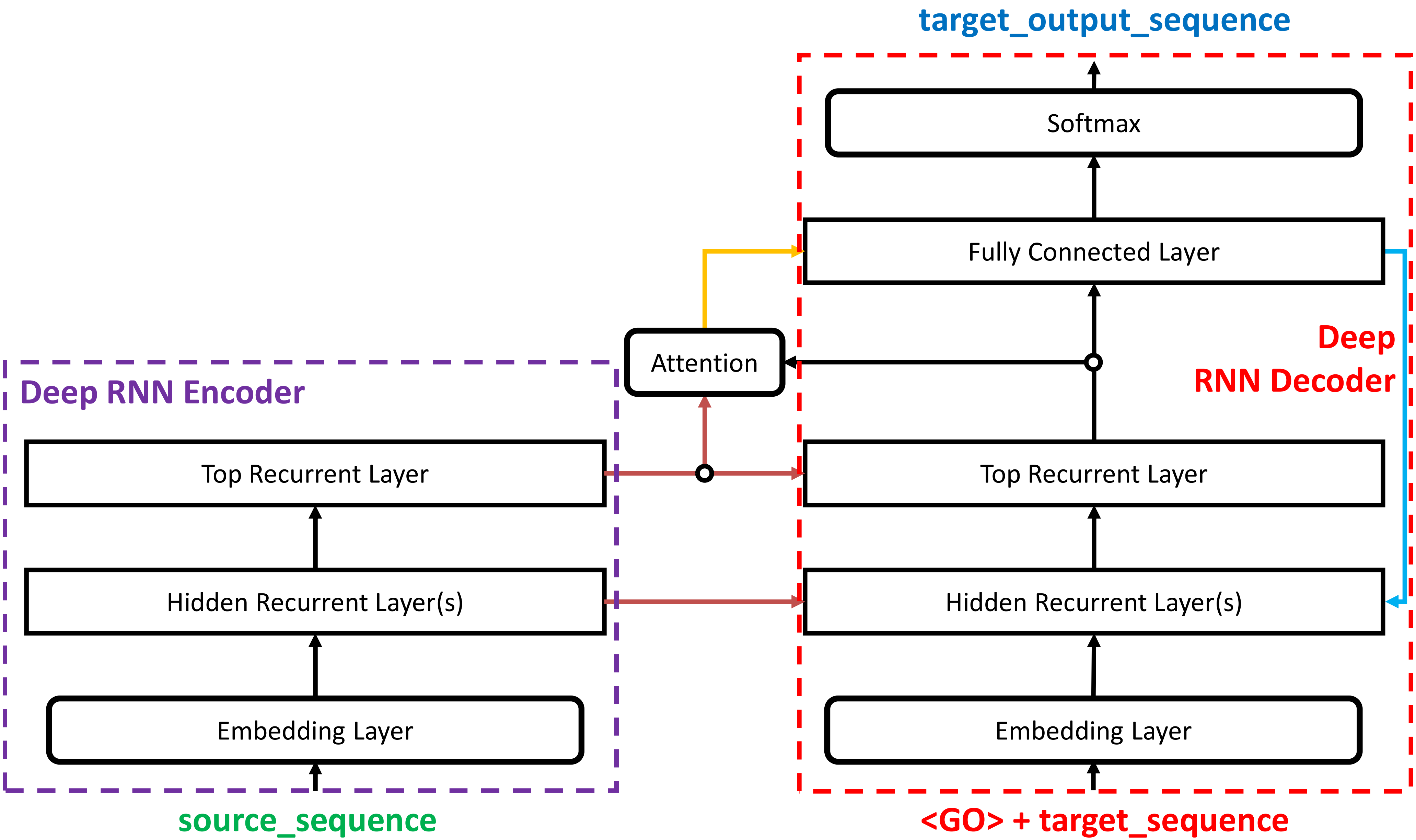
Benefits of Attention Structure

- Significantly improves performance (in many applications)
  - it's very useful to allow the decoder to focus on certain parts of the source
- Solves the bottleneck issue
  - attention allows decoder to look directly at the source (and "bypass" the bottleneck)
- Helps with vanishing gradient problem
  - provides shortcut to far away states
- provides some interpretability
  - inspecting attention distribution we can see what the decoder was focusing on

Deep RNN Enc-Dec with Attention



Deep RNN Enc-Dec with Attention



Deep RNN Enc-Dec with Attention