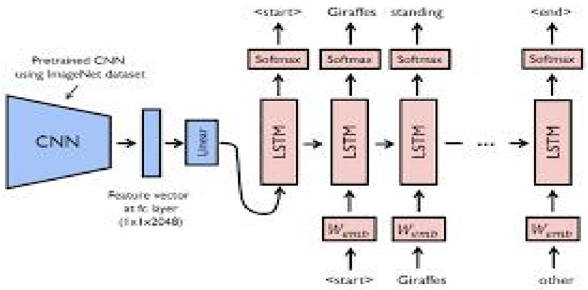
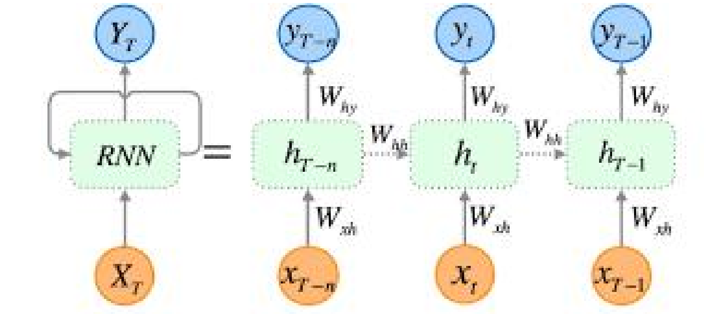
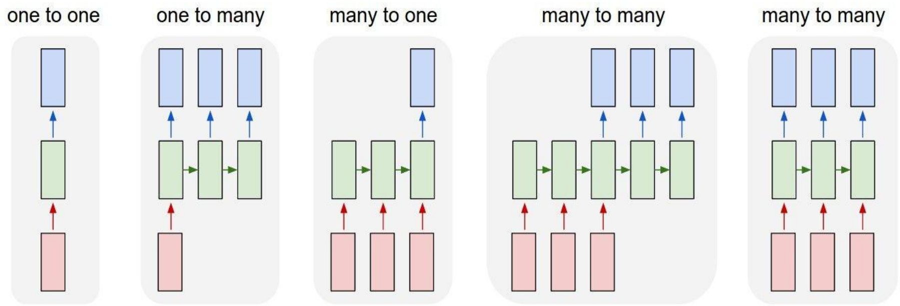
Chapter-3-part-II
Recurrent Neural Networks

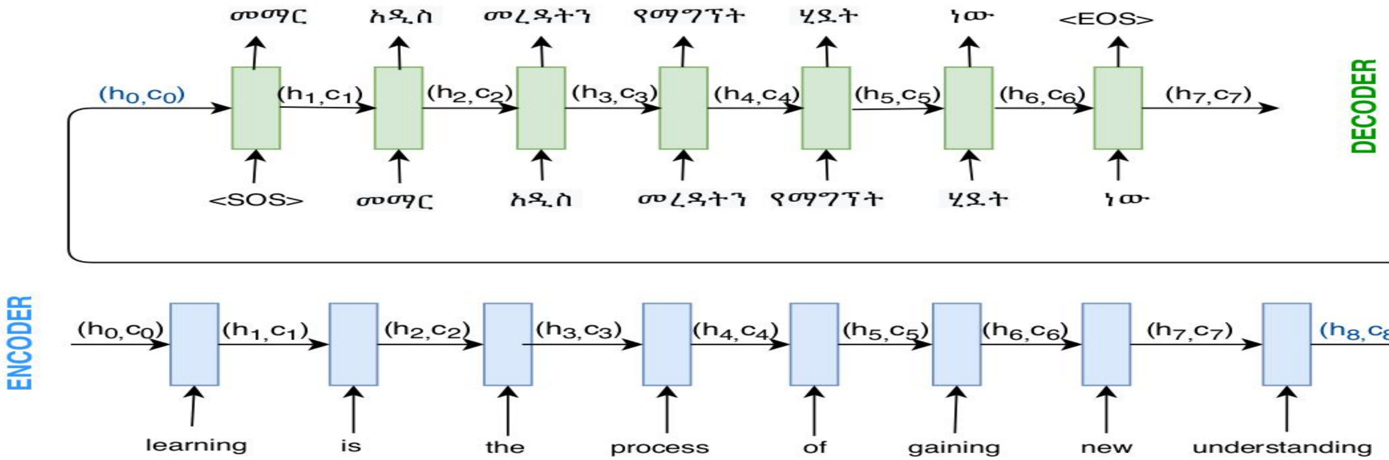
https://www.geeksforgeeks.org/ml-back-propagation-through-time/

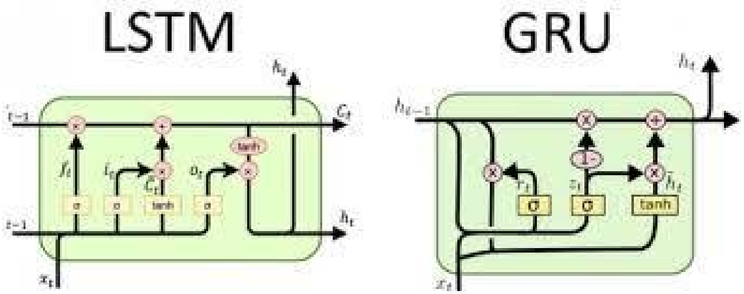


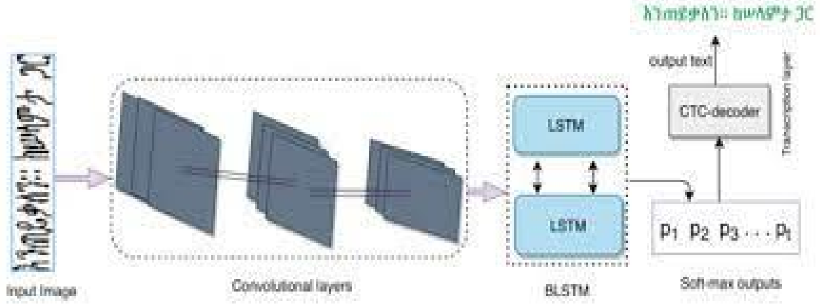


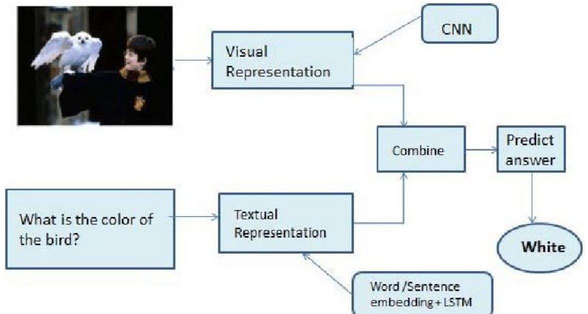
1 Standard RNN architecture and an unfolded structure with T time











What is Recurrent Neural Networks?

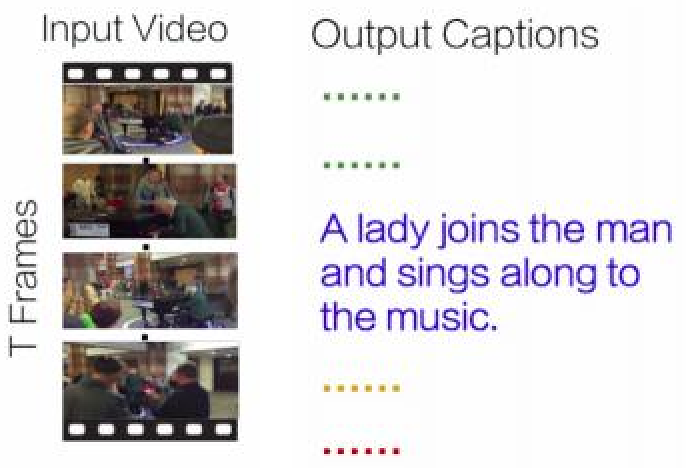
Why existing convnets are insufficient?

Variable sequence length inputs and outputs!

Example task: video captioning

Input video can have variable number of frames

Output captions can be variable length.



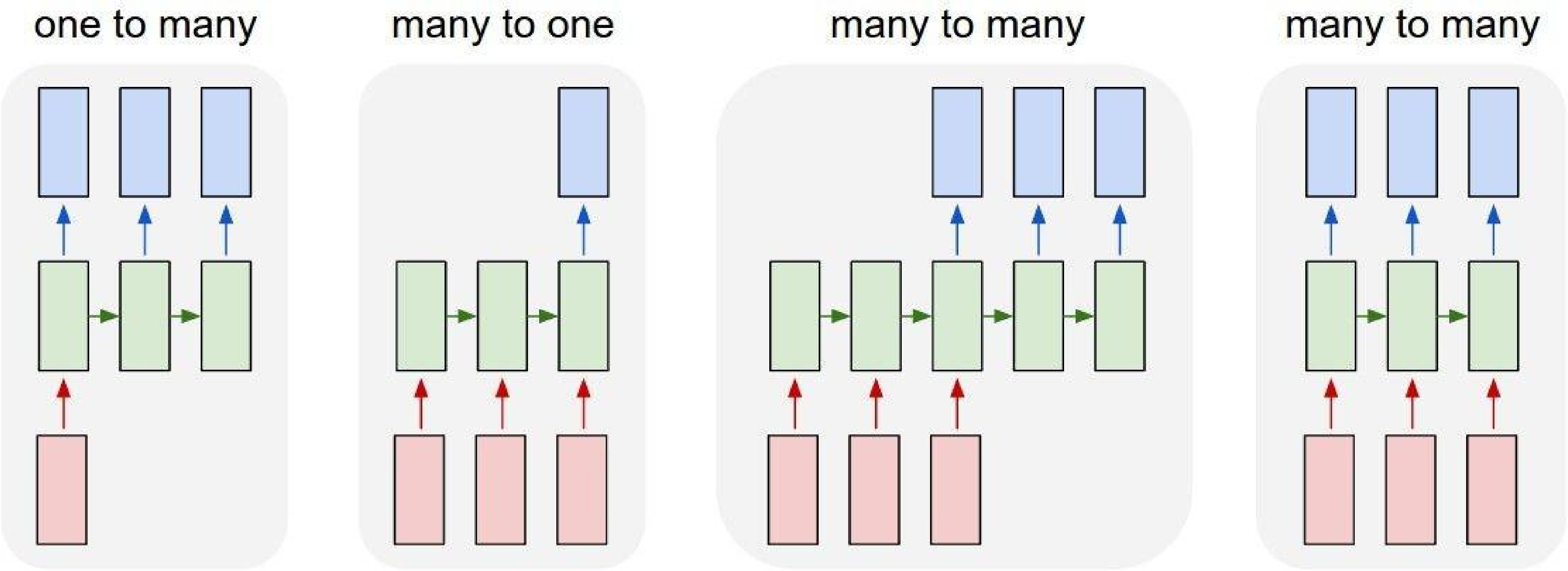
What is Recurrent Neural Networks?

Motivation

- In artificial intelligence applications we often deal with SEQUENCES (e.g. speech, video, machine translation…)
- We need models that take information of SEQUENCE

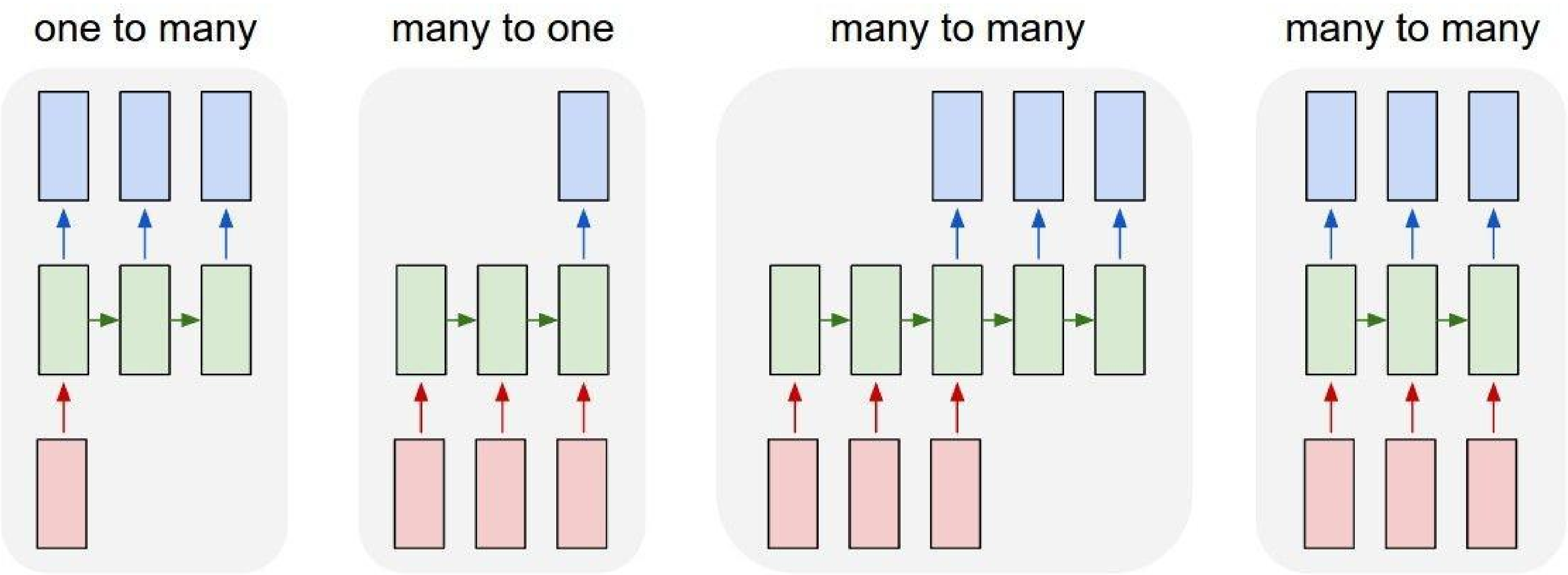
- HOW?
- By recurrency

Recurrent Neural Networks: Process Sequences



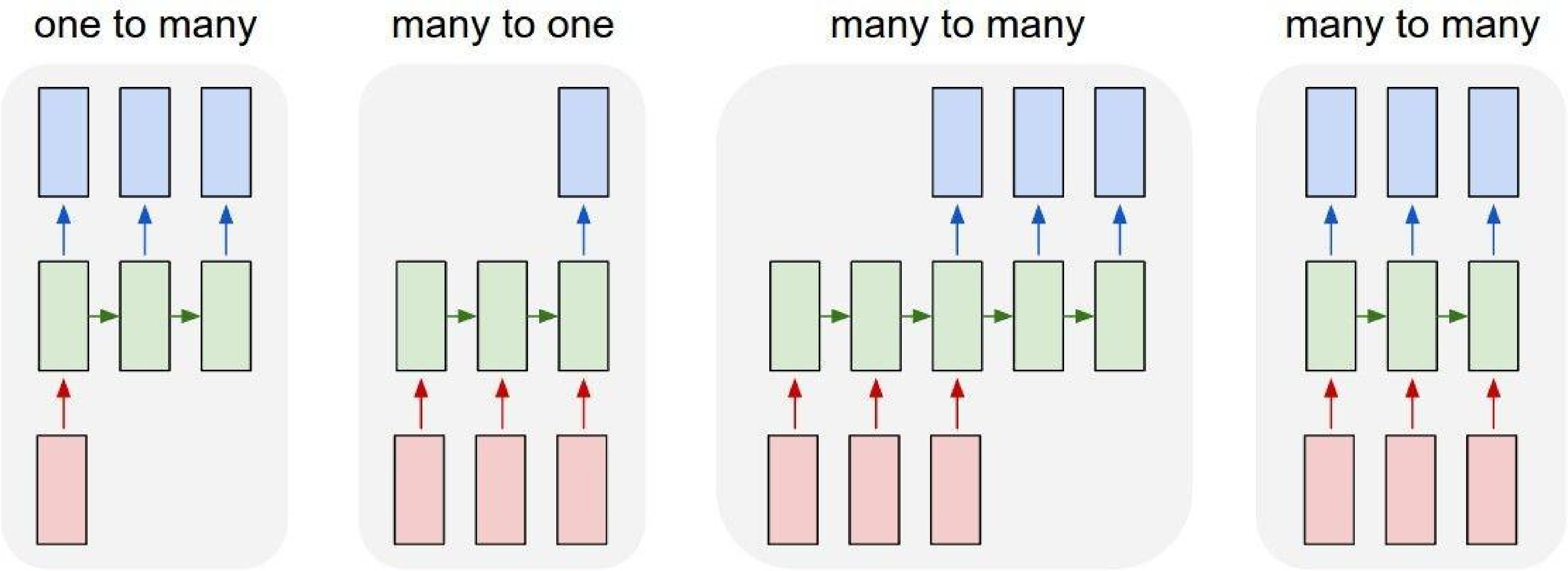
e.g. **Image Captioning**
image -> sequence of words

Recurrent Neural Networks: Process Sequences



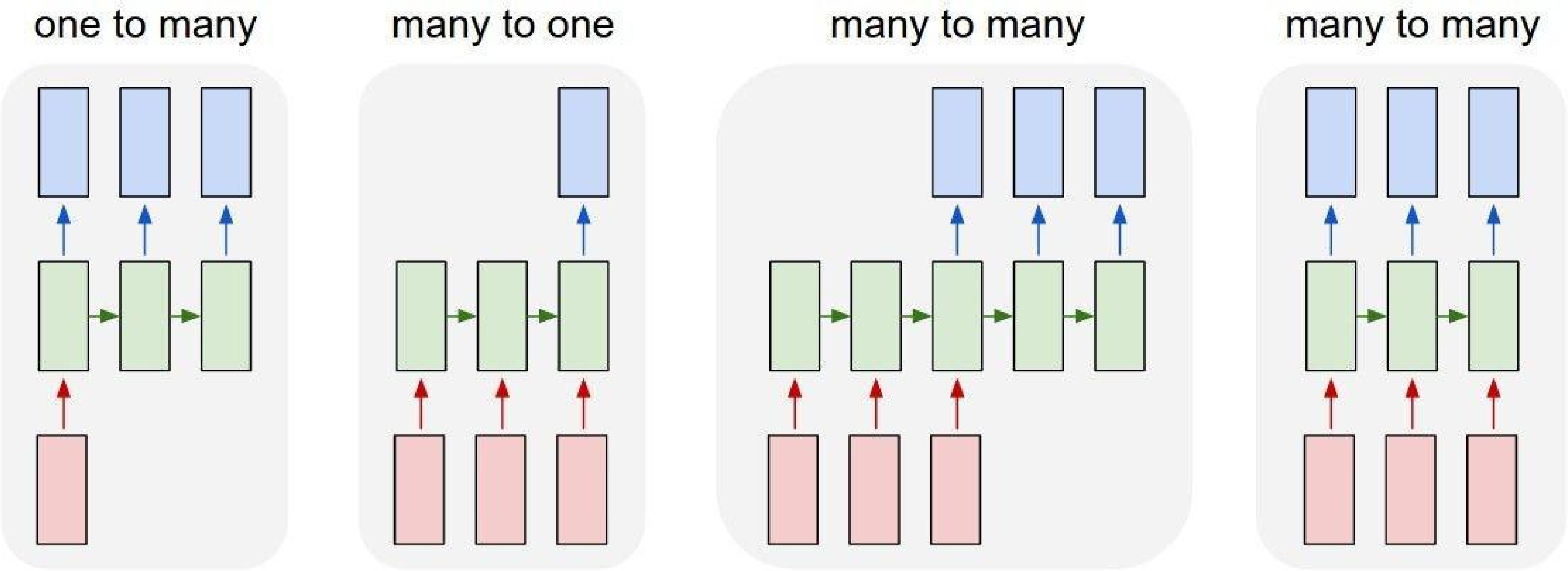
e.g. **Sentiment Classification**
sequence of words -> sentiment

Recurrent Neural Networks: Process Sequences



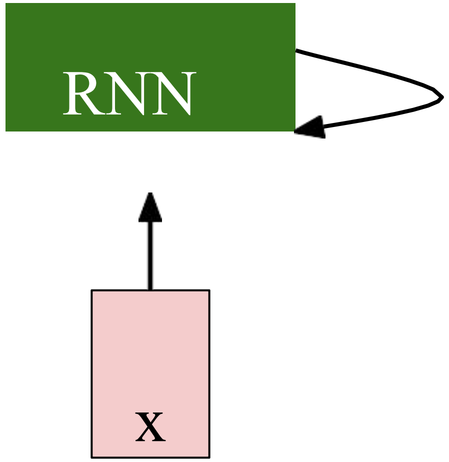
e.g. **Machine Translation**
seq of words -> seq of words

Recurrent Neural Networks: Process Sequences

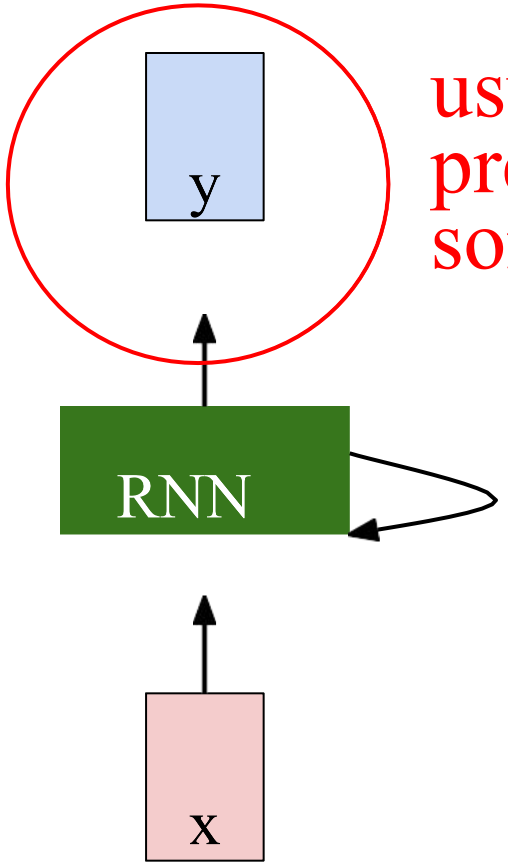


e.g. **Video classification on frame level**

Recurrent Neural Network



Recurrent Neural Network



usually want to predict a vector at some time steps

Recurrent Neural Network...

We can process a sequence of vectors **x** by applying a **recurrence formula** at every time step:

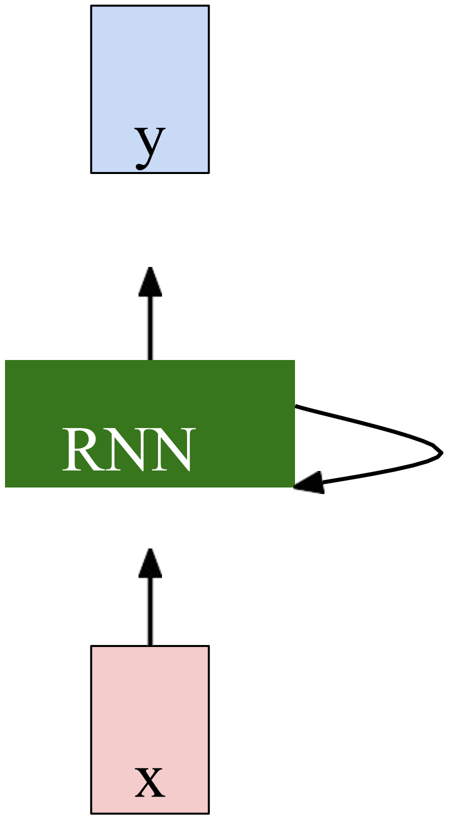
$$\boxed{h_t} = \boxed{f_W}(\boxed{h_{t-1}}, \boxed{x_t})$$

new state

old state

input vector at some time step

some function with parameters W

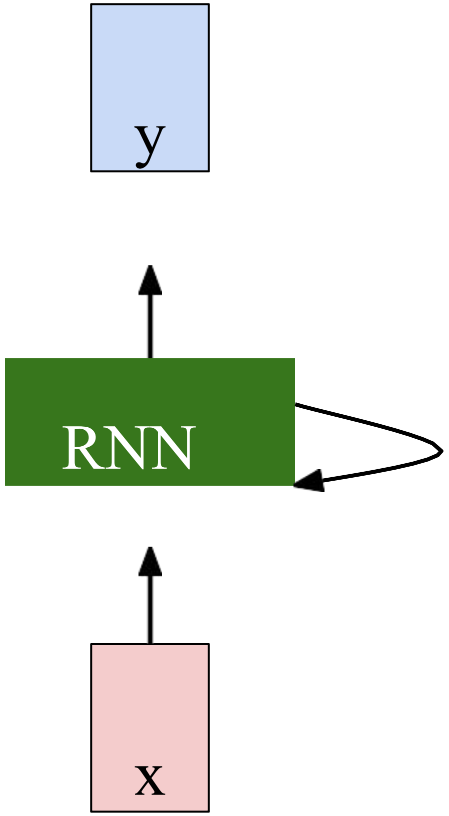


Recurrent Neural Network...

We can process a sequence of vectors **x** by applying a **recurrence formula** at every time step:

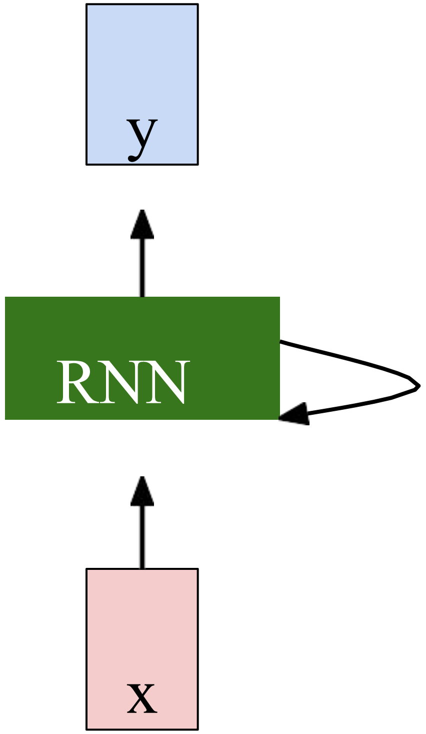
$$h_t = f_W(h_{t-1}, x_t)$$

Notice: the same function and the same set of parameters are used at every time step.



(Vanilla) Recurrent Neural Network

The state consists of a single *“hidden”* vector **h**:

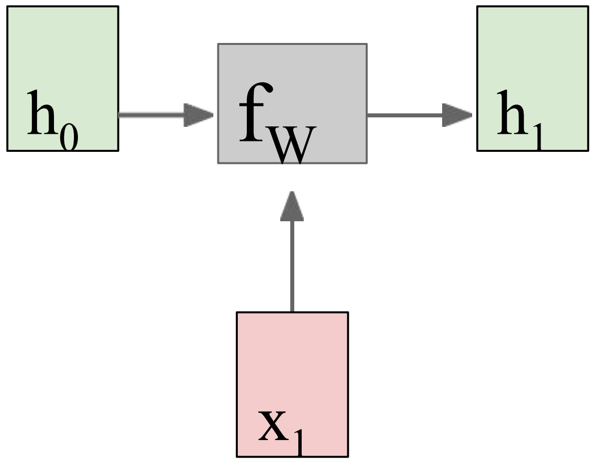


$$h_t = f_W(h_{t-1}, x_t)$$

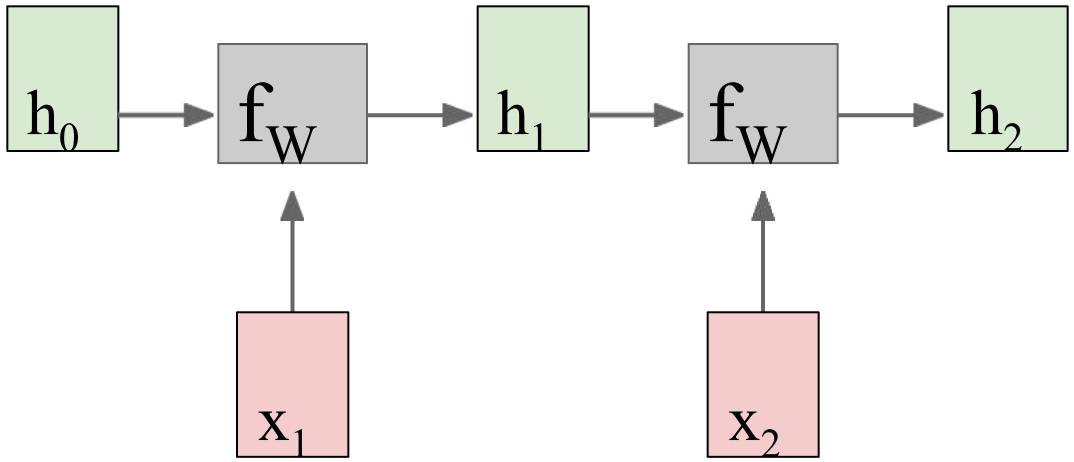
$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

$$y_t = W_{hy}h_t$$

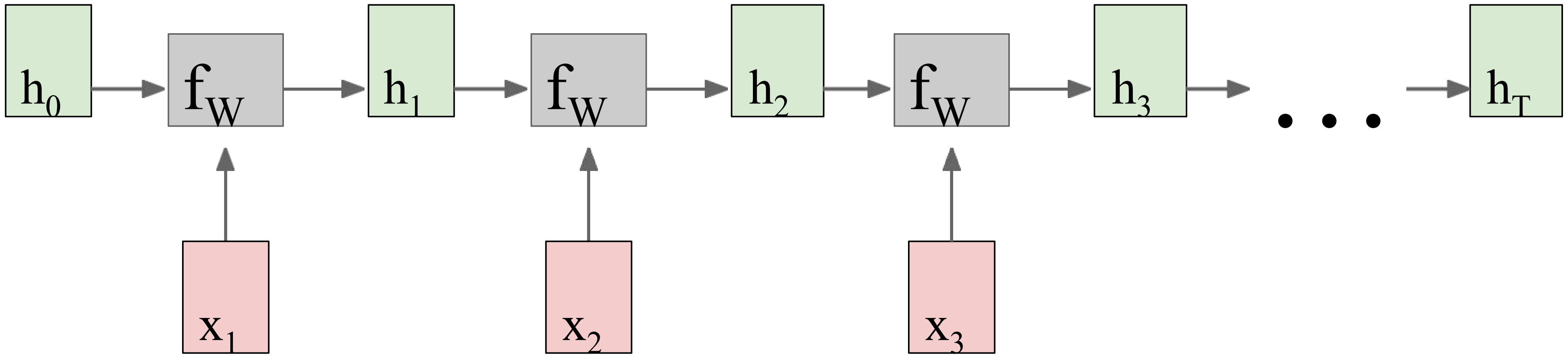
RNN: Computational Graph



RNN: Computational Graph

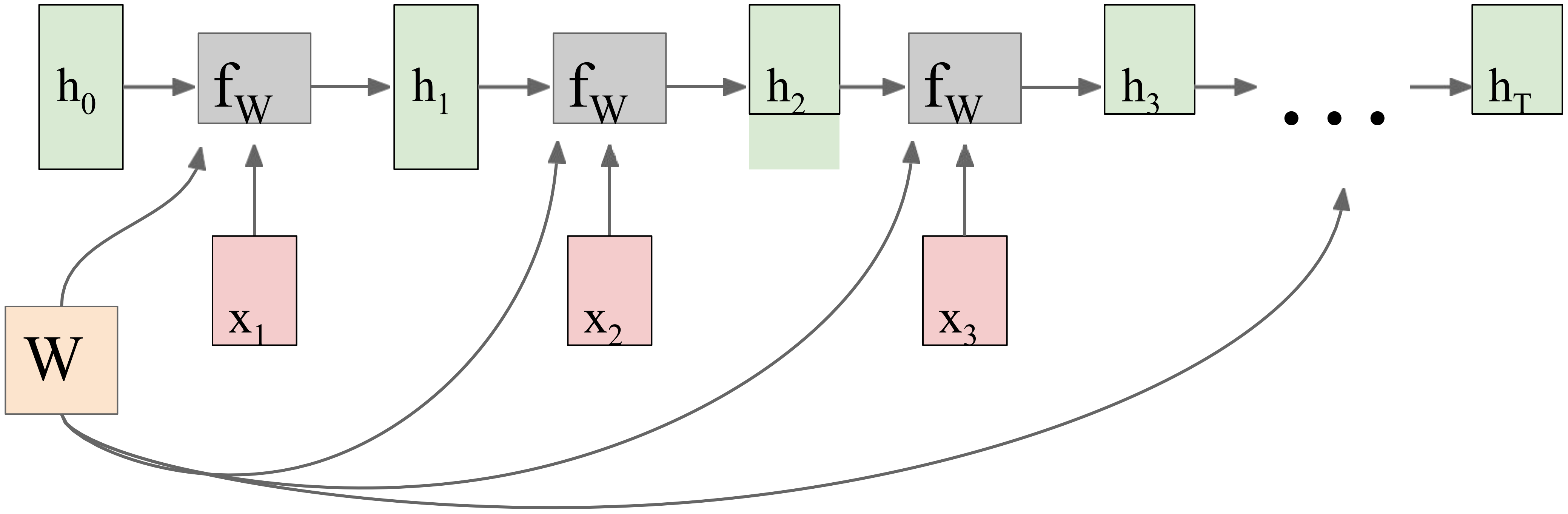


RNN: Computational Graph

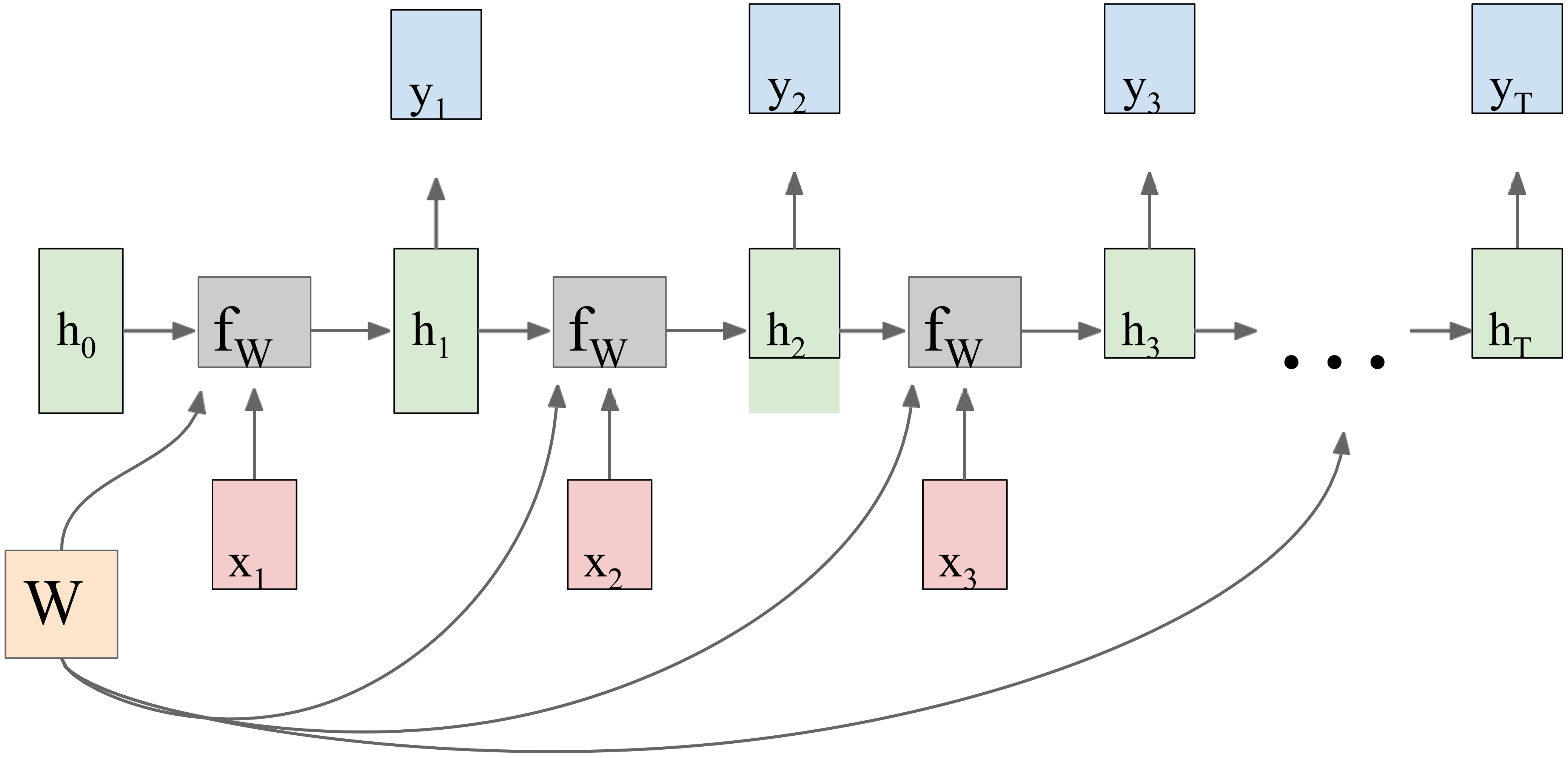


RNN: Computational Graph

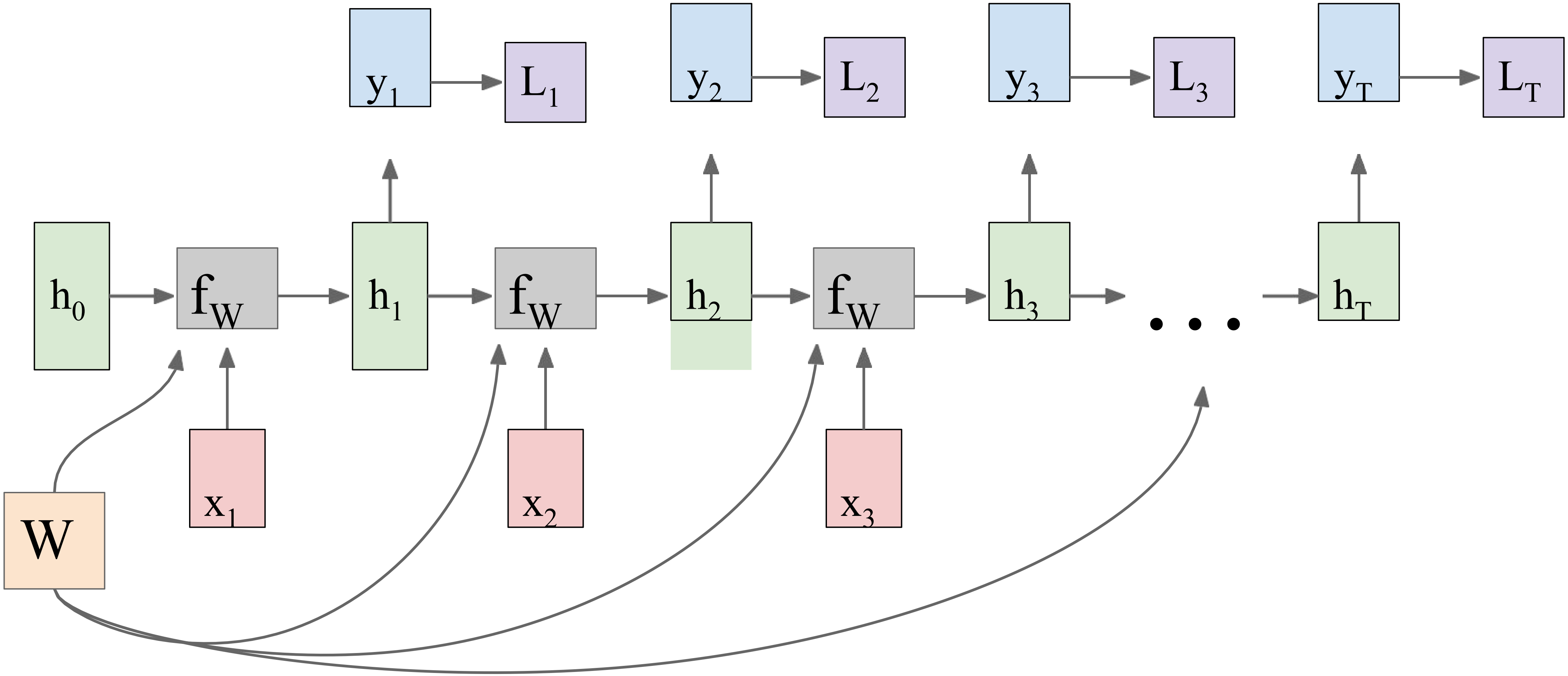
Re-use the same weight matrix at every time-step



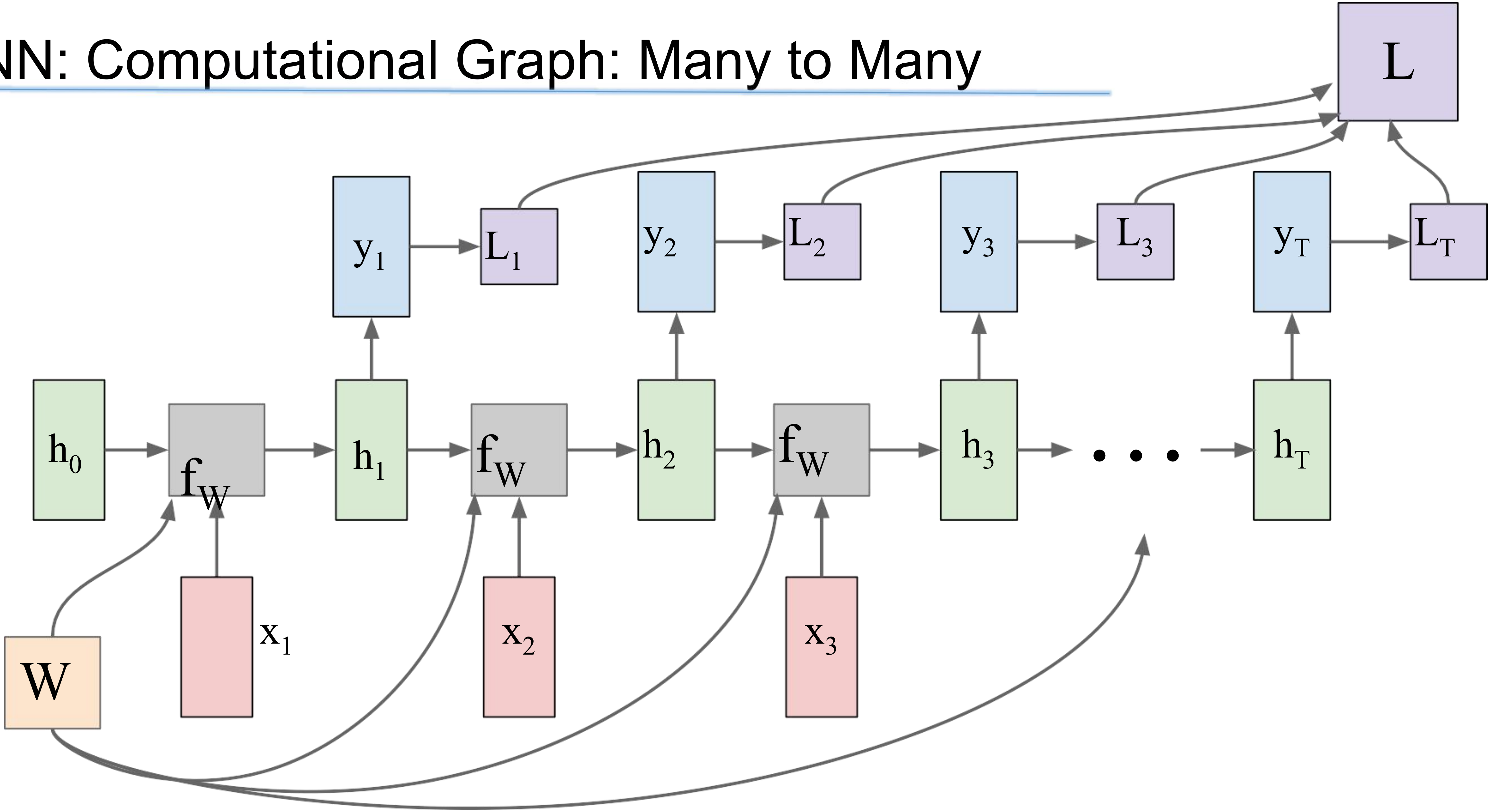
RNN: Computational Graph: Many to Many



RNN: Computational Graph: Many to Many

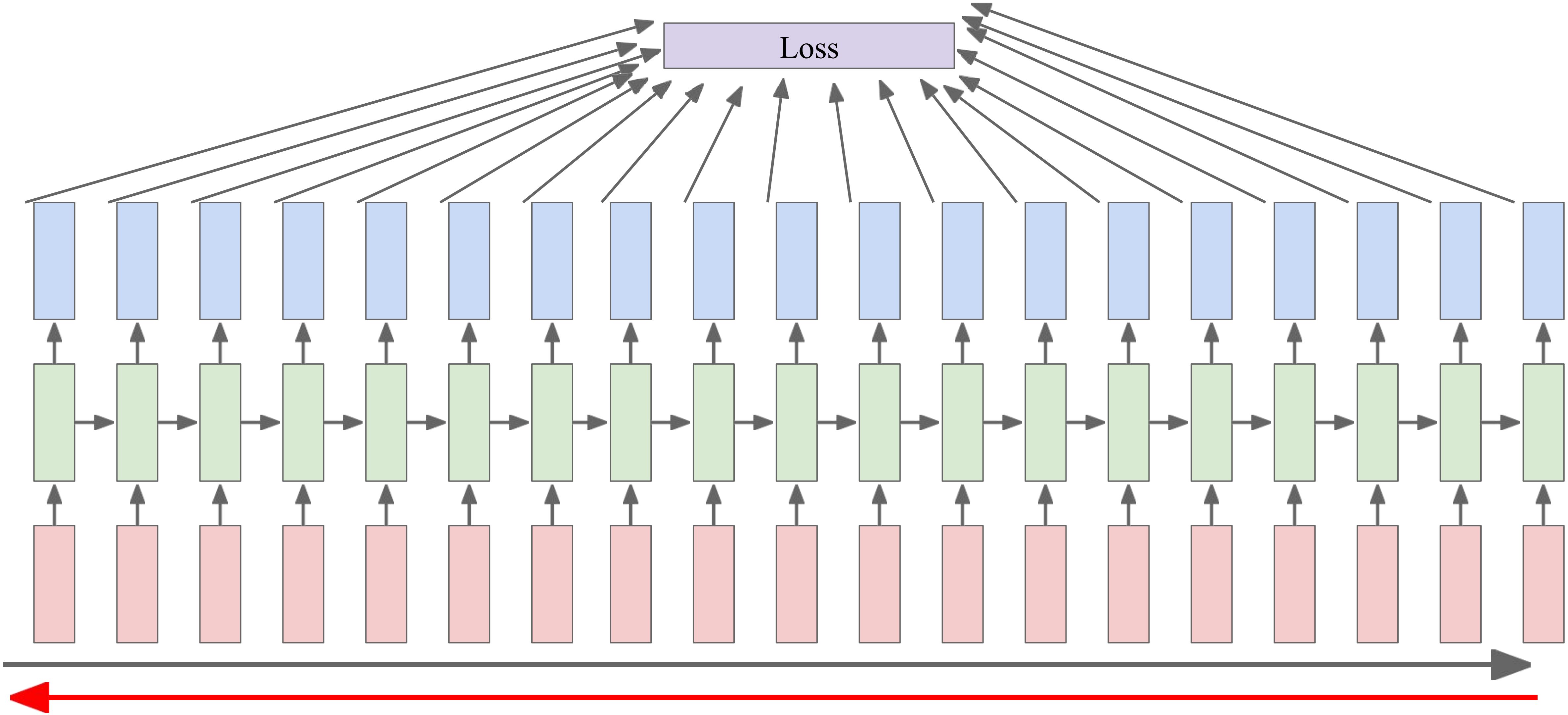


RNN: Computational Graph: Many to Many



Backpropagation through time

Forward through entire sequence to compute loss, then backward through entire sequence to compute gradient



Training a RNN I: BPTT

- Backpropagation through time (BPTT): The training algorithm for updating network weights to minimize error including time

Remember BackPropagation

1. Present a training input pattern and propagate it through the network to get an output.

1. Compare the predicted outputs to the expected outputs and calculate the error.
2. Calculate the derivatives of the error with respect to the network weights.
3. Adjust the weights to minimize the error.
4. Repeat.

BPTT I: Loss

$$
\begin{aligned}
& L\left(\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(\tau)}\}, \{\boldsymbol{y}^{(1)}, \ldots, \boldsymbol{y}^{(\tau)}\}\right) \\
=& \sum_t L^{(t)} \\
=& -\sum_t \log p_{\text{model}}\left(y^{(t)} \mid \{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(t)}\}\right),
\end{aligned}
$$

The total loss for a given sequence x(t)

$$
\frac{\partial L}{\partial L^{(t)}} = 1.
$$

Our goal is to calculate the gradients of the error with respect to our parameters U, W and V and and then learn good parameters using Stochastic Gradient Descent.

Ref: https://www.geeksforgeeks.org/ml-back-propagation-through-time/

Vanishing/ Exploding gradients

sources: https://machinelearningmastery.com/exploding-gradients-in-neural-networks/
https://machinelearningmastery.com/how-to-fix-vanishing-gradients-using-the-rectified-linear-activation-function/

Vanishing/ **Exploding** gradient ...

- During training gradients explode/vanish easily because of depth-in-time → **Exploding**/**Vanishing** gradients!

- **Vanishing Gradient** occurs when the derivative or slope will get **smaller and smaller** as we go backward with every layer during backpropagation.

- **Exploding gradient** occurs when the derivatives or slope will get **larger and larger** as we go backward with every layer during backpropagation. This situation is the exact opposite of the vanishing gradients.

Vanishing/ **Exploding** gradient...

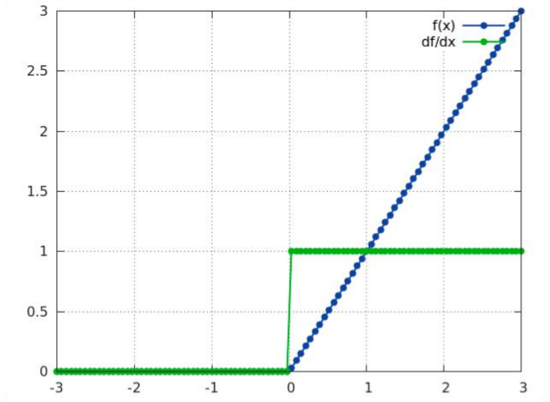
- A **vanishing** Gradient problem occurs with the sigmoid and tanh activation function because the derivatives of these units are between 0 to 0.25 and 0 to 1 respectively.
- Therefore, the updated weight values are small, and the new weight values are very similar to the old weight values.
- A **Exploding** Gradient problem happens because of weights, not because of the activation function.
- Due to high weight values, the derivatives will also higher so that the new weight varies a lot to the older weight, and the gradient will never converge.

Question

- Do FNNs with several hidden layers suffer from vanishing gradient problem?

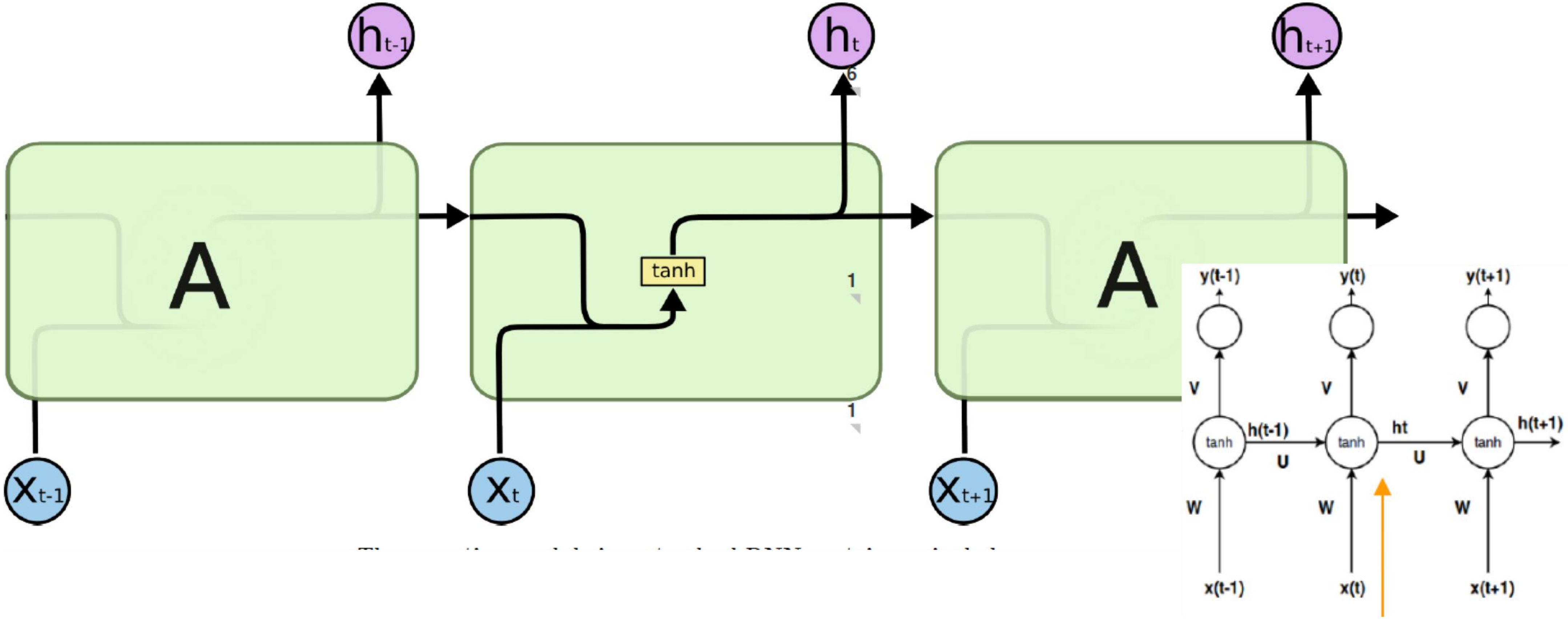
Standard Solutions

- Proper initialization of Weight Matrix
- Regularization of outputs or Dropout
- Use of ReLU Activations as it’s derivative is either 0 or 1



Gated Units

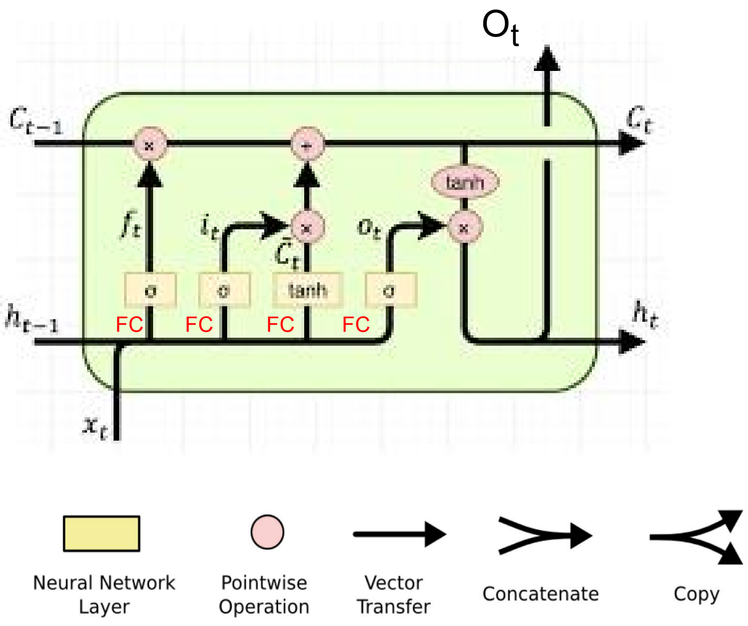
Standard RNN



SLIDES FROM: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Long-Short Term Memory (LSTM)

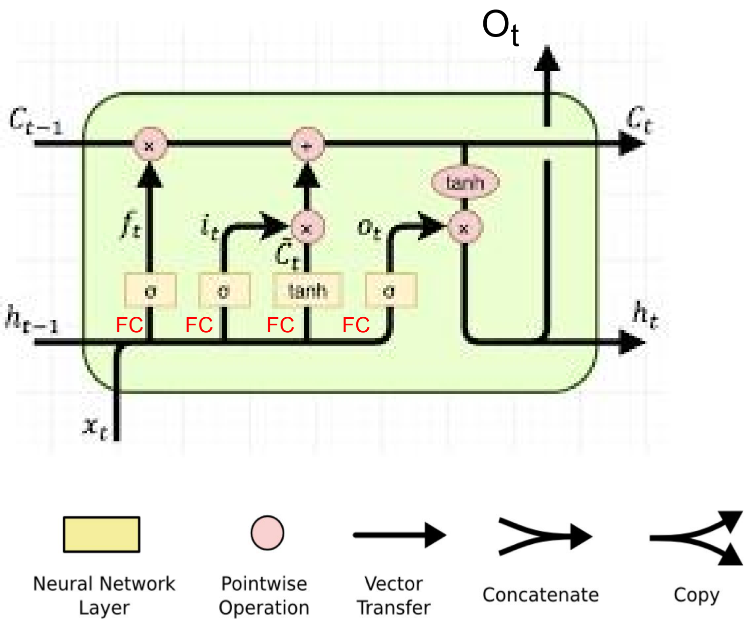
- LSTM:
  - an improvement over RNN and used for modeling complex sequential data.
  - has three inputs (C_{t-1}, h_{t-1}, X_t) and three outputs(C_t, h_t, O_t), where O_t is the softmax of h_t.
  - **gating** mechanism: that controls the memoizing process.
  - **gates** are FC leyer followed an activation function:
    - forget gate
    - input gate
    - cell-state gate
    - output gate
  - **cell state**: encode a kind of aggregation of data from all previous time-steps that have been processed,
  - **hidden state:** is meant to encode a kind of characterization of the most recent time-step.

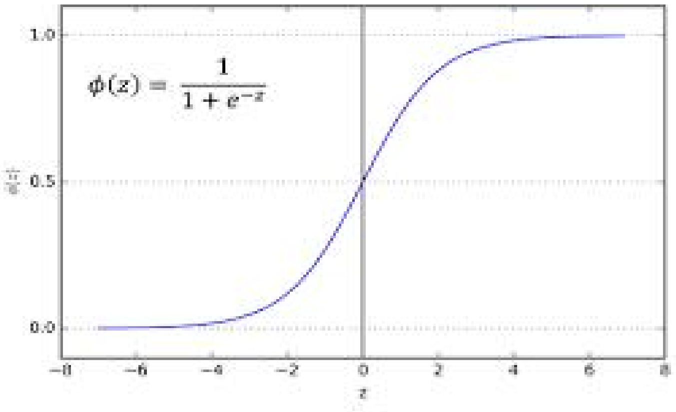


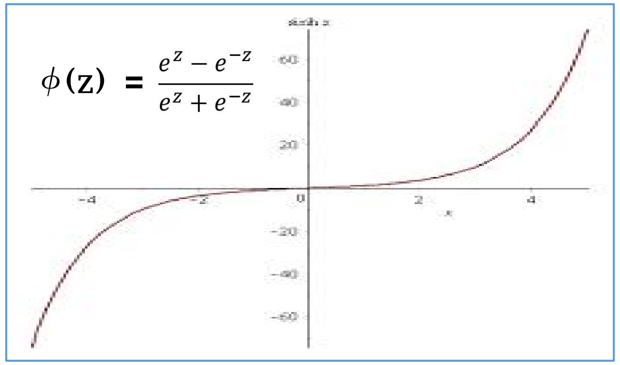
Schmidhuber, Jürgen, and Sepp Hochreiter. "Long short-term memory." Neural Comput 9, no. 8 (1997): 1735-1780.

Long-Short Term Memory (LSTM)...

- LSTM:
  - consists of:
    - five activation functions (**sigmoid** or **tanh**), and
    - four math operations (3 multi, 1 add)



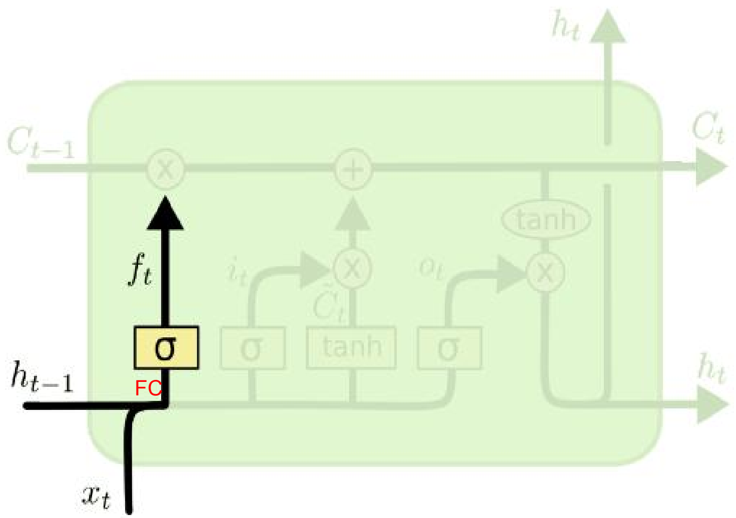




Schmidhuber, Jürgen, and Sepp Hochreiter. "Long short-term memory." Neural Comput 9, no. 8 (1997): 1735-1780.

Long-Short Term Memory (LSTM)...

Forget Gate



What part of memory to "forget" – zero means forget this bit

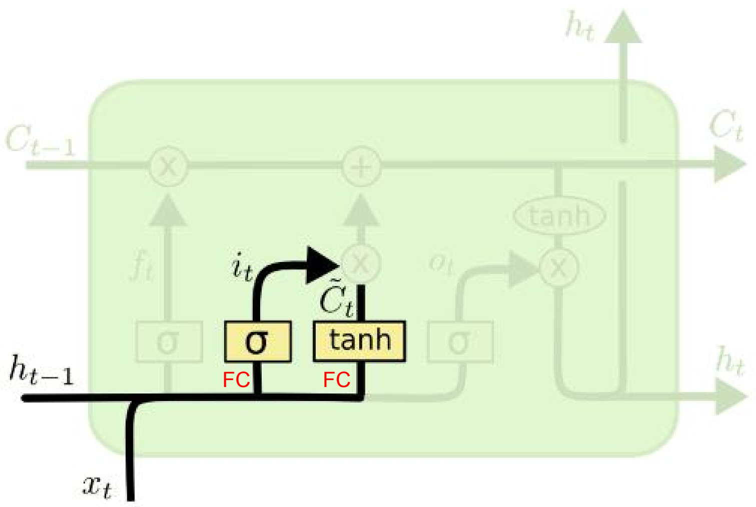
Forget Gate:

$$f_t = \sigma\left(W_f \cdot \underbrace{[h_{t-1}, x_t]}_{\text{Concatenate}} + b_f\right)$$

- Previous hidden-state and current input are concatenated pass through FC and a sigmoid activation function.

Long-Short Term Memory (LSTM)...

Input Gate



What bits to insert into the next states

Input Gate Layer

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

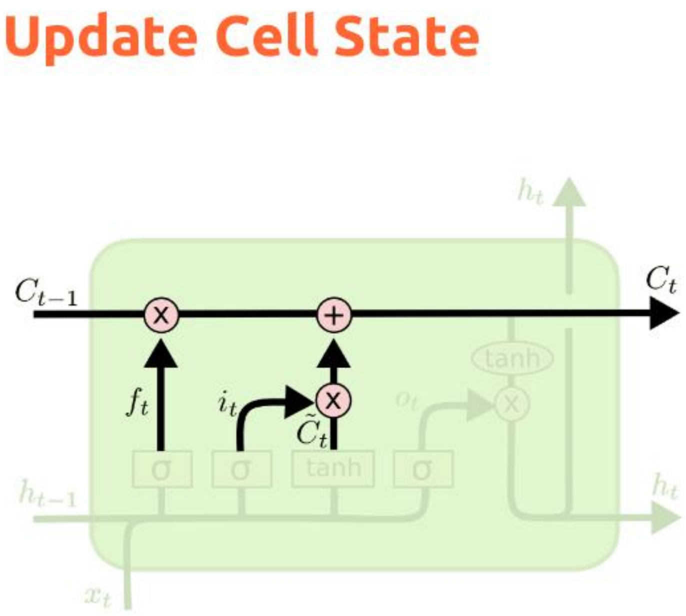
New contribution to cell state

$$\tilde{C}_t = \tanh(\underbrace{W_C \cdot [h_{t-1}, x_t] \ + \ b_C}_{\text{Classic neuron}})$$

What content to store into the next state

- Previous hidden state and current input are **concatenated**, and are passed through two parallel branches:
    - the first branch has a FC network, followed by a sigmoid,
    - second branch has a FC network, followed by a tanh function.

Long-Short Term Memory (LSTM)...



Next memory cell content – mixture of not-forgotten part of previous cell and insertion

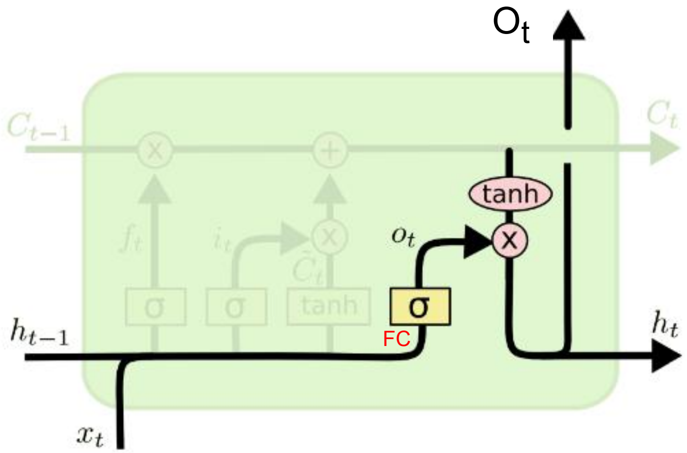
Update Cell State (memory):

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

- The output vectors of two parallel branches are multiplied elementwise.
- The output after multiplication is added to the cell state after the cell state is updated with the forget gate.

Long-Short Term Memory (LSTM)...

Output Gate



What part of cell to output

Output Gate Layer

$$o_t = \sigma\left(W_o \left[h_{t-1}, x_t\right] + b_o\right)$$

Output to next layer

$$h_t = o_t * \tanh\left(C_t\right)$$

- The current input is concatenated with the hidden state and passed through the Fully connected network
- once appling the sigmoid, and this output is multiplied element-wise with cell state after it passes through a tanh function.
- The result of this multiplication is the current output/hidden state.

How to compute parameter LSTM?

Hint:

- we have four FC networks at different places in the LSTM.
- each FC network has an input layer and an output layer.
- the **number of neurons** in each FC network input and output is dependent on the dimension of the input vector(m) and cell state vector(n).
- Since the input to all FC networks is a concatenated vector of current input and hidden state vector(ht-1 + Xt). The input layer of all the FC networks has (m+n) neurons.
- Since the output of the FC network will be multiplied or added to the cell state element-wise, the output layer of FC network has a number of neutrons equal to the dimension of the cell state(n).

How to compute parameter LSTM?

Hint:

- we have four FC networks at different places in the LSTM.
- each FC network has an input layer and an output layer.
- the **number of neurons** in each FC network input and output is dependent on the dimension of the input vector(m) and cell state vector(n).
- Since the output of the FC network will be multiplied or added to the cell state element-wise, the output layer of FC network has a number of neutrons equal to the dimension of the cell state(n).
- To calculate number of parameters. Each FC network has a parameter weight matrix of [(**m**+**n**)***n**] and a bias values of 'n'.
- Total parameters at each FC network **(m*n + sqr(n) + n)** and for four FC networks it will be **4*(m*n + sqr(n) + n)**.

Why do we need a cell state when we have a hidden state of LSTM?

Hint: Vanishing/exploding gradient issues.

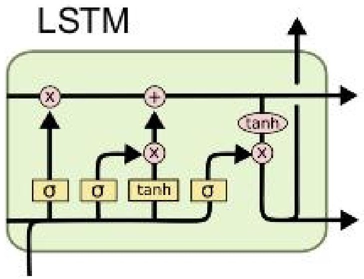
- if no cell-state: the hidden state feature vector will pass through the Neural network and gets to interact with weight matrices on the way.

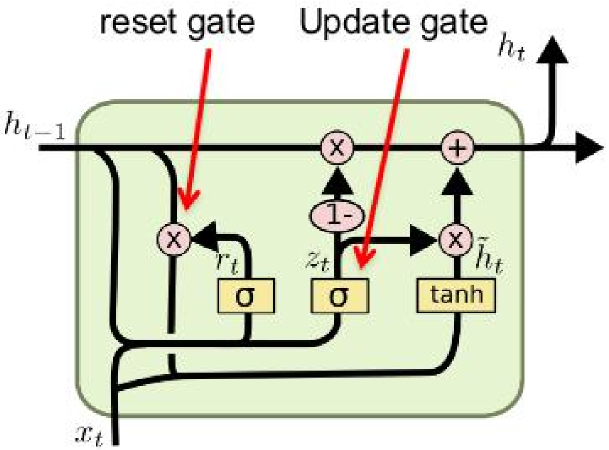
- then this cell-state acts as a **highway without touching** any fully connected layers on the way, it is literally similar to the residual block in a resent architecture.

GRU

GRU – gated recurrent unit

(more compression)





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

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

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

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

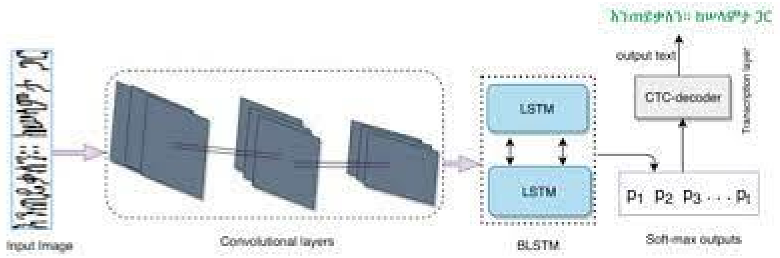
X,*: element-wise multiply

It combines the forget and input into a single update gate.
It also merges the cell state and hidden state. This is simpler than LSTM. There are many other variants too.

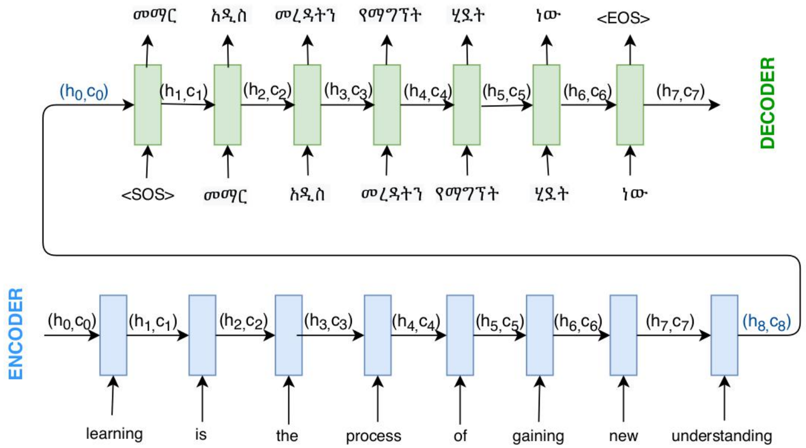
Reading assignment on GRU...?

Open research Topics [CNN, RNN, CNN+RNN]

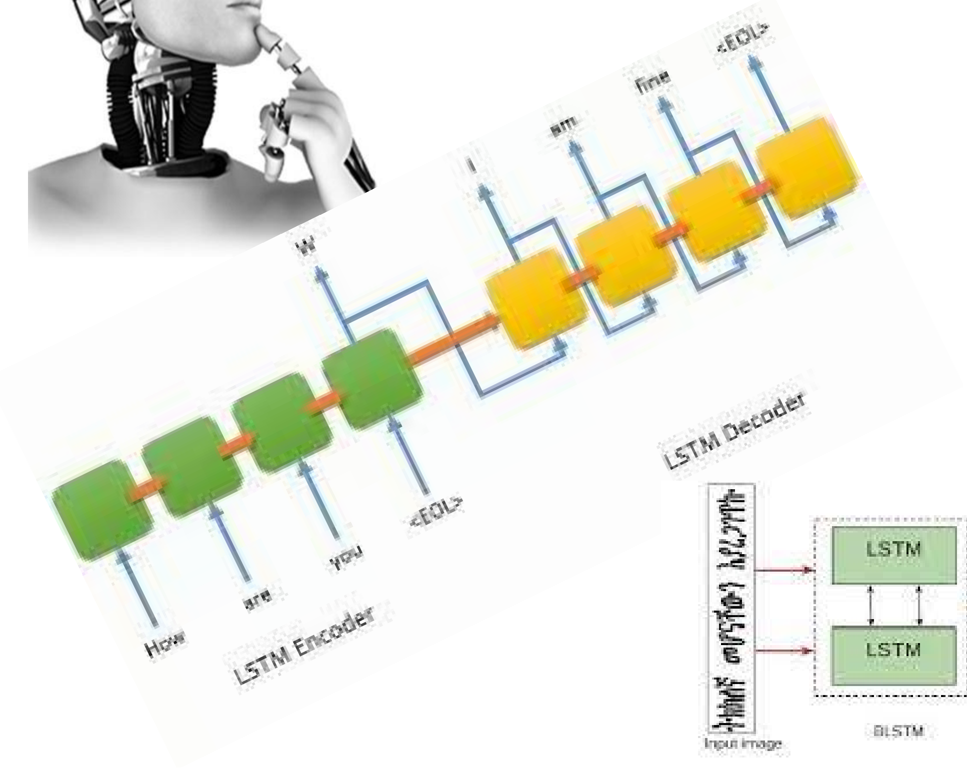
- Speech to text and via for low-resource languages [langues have no enough written documents]
- Visual question answering
- Image captioning
- Language identification from document images [e.g historical documents]
- Ethiopian sign language translation and recogntion
- Agricultural production quality grading and identification (e.g banana coffee, barley etc)
- Multi-modal data analysis(e.g taking the image and medical notes, prescription for disease diagnosis)
- Covid-19...impact prediction in a certain domain (education, economy, health sectors...)
- Text detection and recognition in the wild



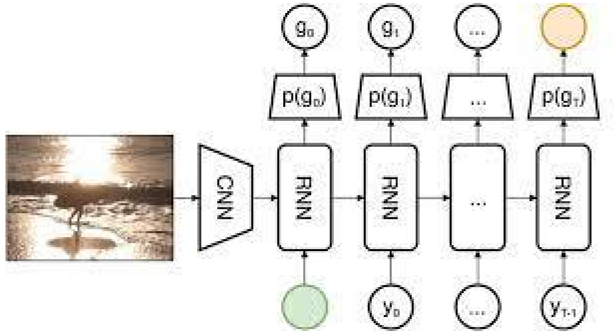
output text
CTC-decoder
Transcription layer
LSTM
LSTM
P1 P2 P3 · · · Pt
Input Image
Convolutional layers
BLSTM
Soft-max outputs



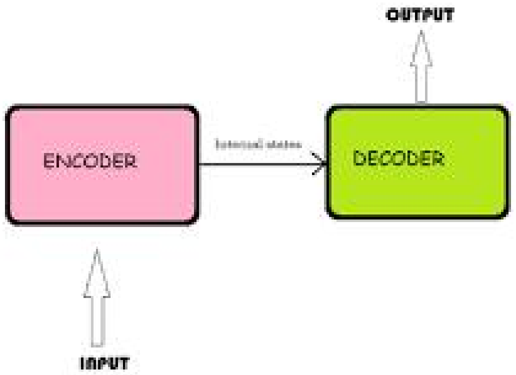
መማር
አዲስ
መረዳትን
የማግኘት
ሂደት
ነው
<EOS>
DECODER
<SOS>
ENCODER
learning
is
the
process
of
gaining
new
understanding



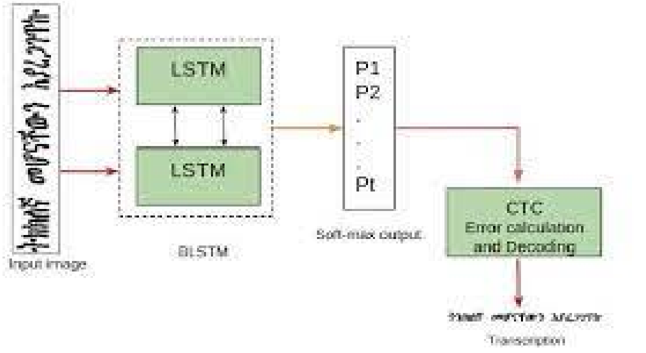
<EOL>
fine
am
I
W
How
are
you
<EOL>
LSTM Encoder
LSTM Decoder



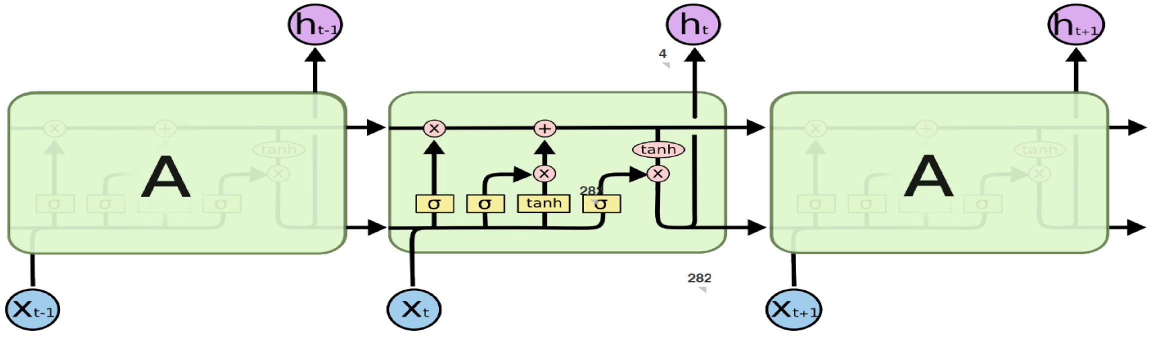
CNN
RNN
RNN
RNN



OUTPUT
ENCODER
DECODER
INPUT



LSTM
LSTM
P1
P2
Pt
CTC
Error calculation
and Decoding
Input image
BLSTM
Soft-max output
Transcription



A
A
tanh
tanh