CGnal

business innovation through algorithms

Recurrent Neural Networks

CGnal S.p.A – Corso Venezia 43 - Milano

14 Dicembre 2022| Milano

Agenda

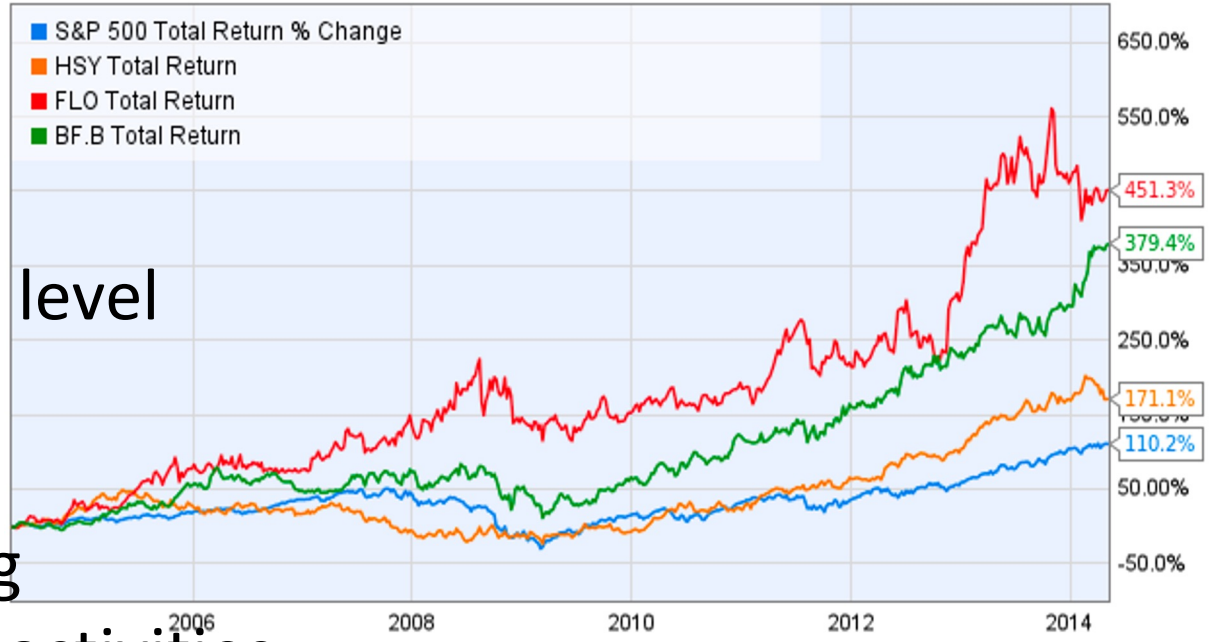
1. Recurrent Neural Networks
2. Lstm & GRUs
3. Implementing RNNs in Keras

Recurrent Neural Networks

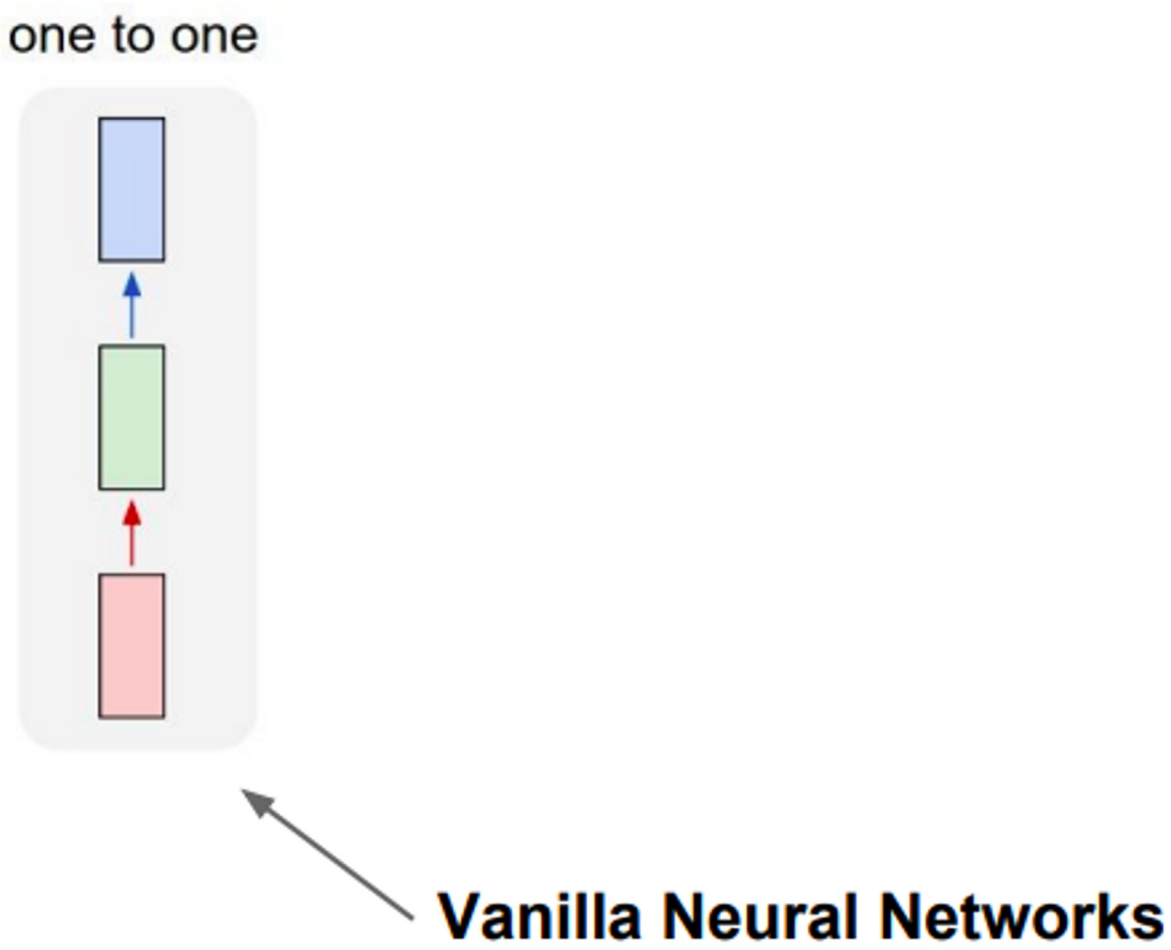
RNNs for Sequence Modeling

Sequential Data

- Time Series
  1. Financial data, stoc market
  2. Health Care : pulse rate, sugar level
- Text and Speech
- Spatiotemporal data:
  1. Self driving and object tracking
  2. Seismic waves : Plate tectonic activities
- Physics:
  1. Anomaly detection, failure analysis etc

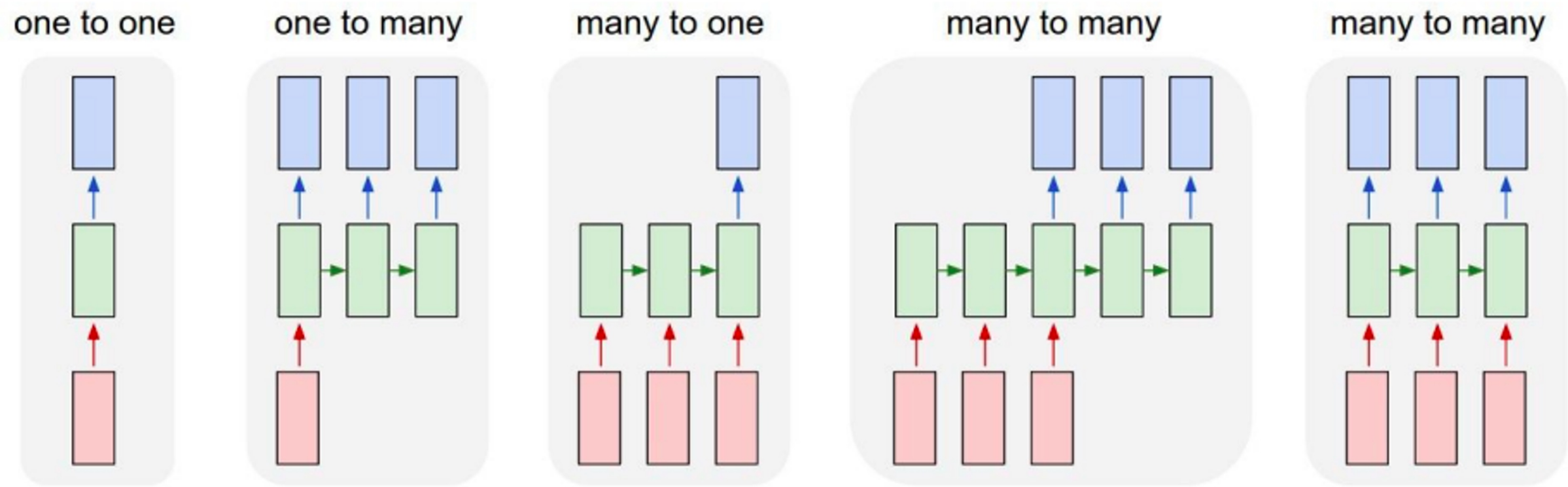


RNNs for Sequence Modeling



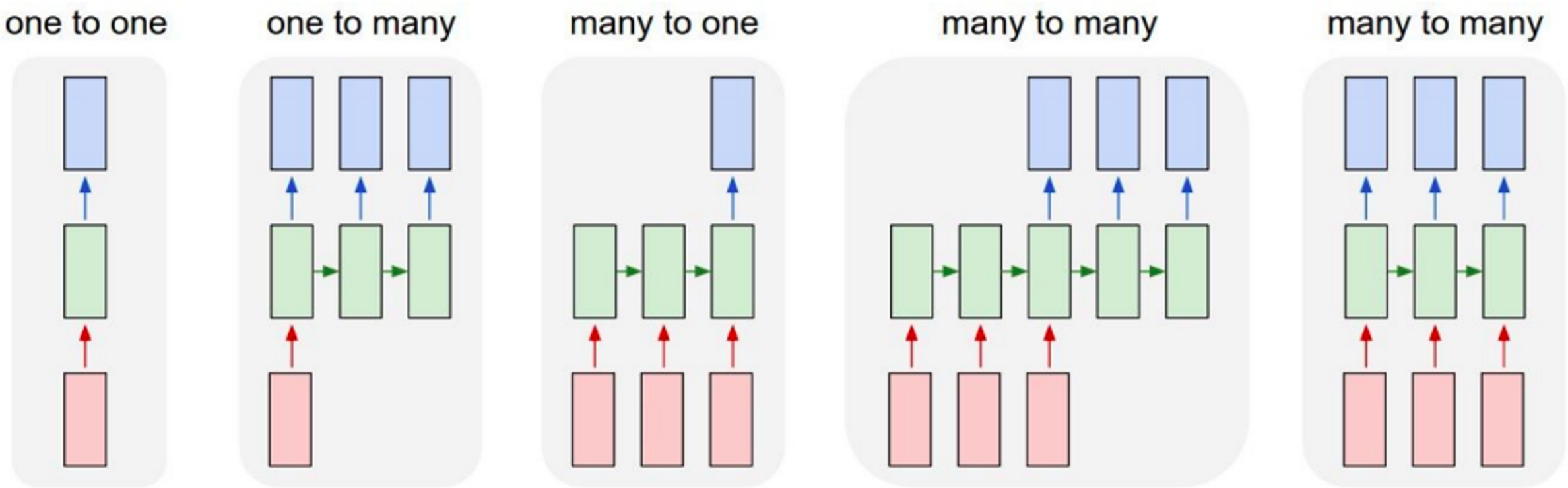
Credits: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture10.pdf

RNNs for Sequence Modeling



Credits: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture10.pdf

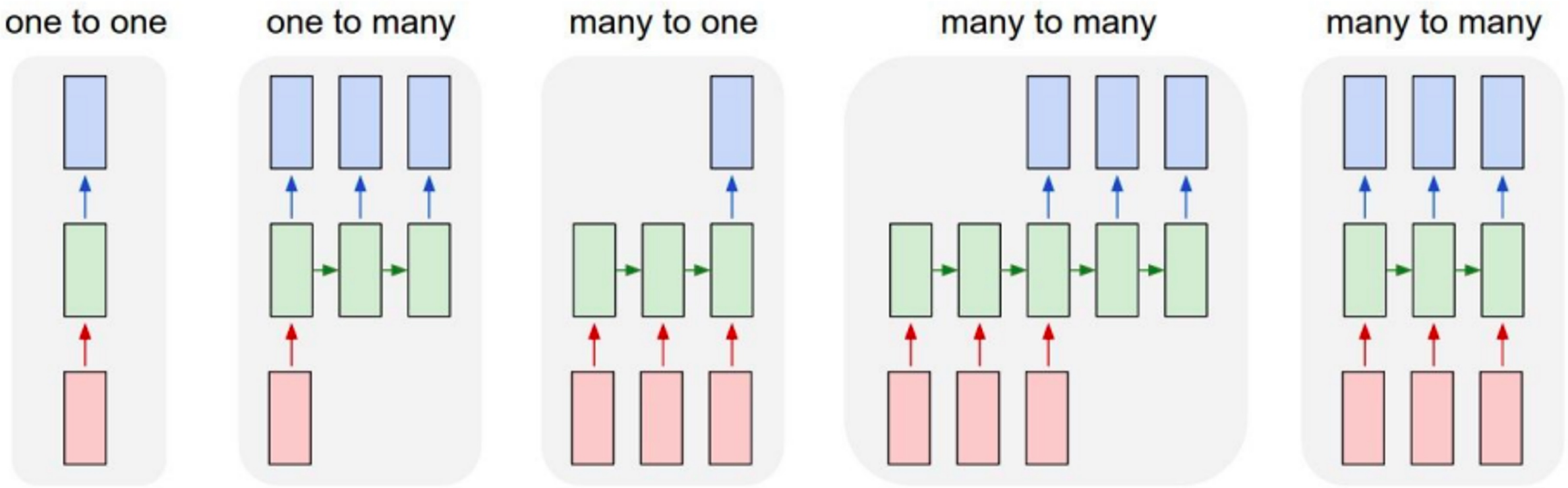
RNNs for Sequence Modeling



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

Credits: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture10.pdf

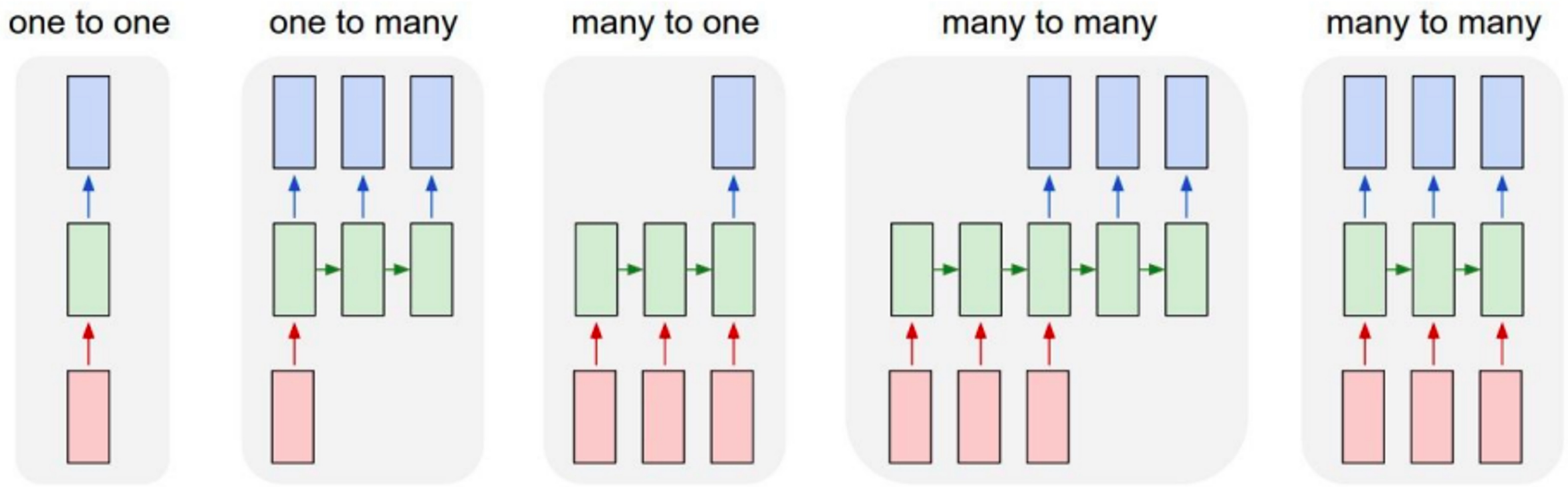
RNNs for Sequence Modeling



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

Credits: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture10.pdf

RNNs for Sequence Modeling



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

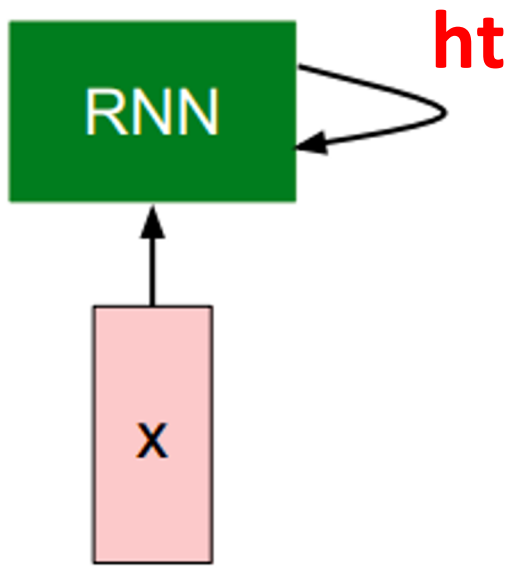
Multivariate timeseries predicton

Credits: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture10.pdf

RNN architecture

Steps :

1. Take input x at time step t1
2. Update hidden state h1 at time step t1
3. For next input x at time step t2, use updated hidden state h1 from previous time step t1 to calculate h2 at timestep t2
4. Keep repeating the above two steps for t1….tN

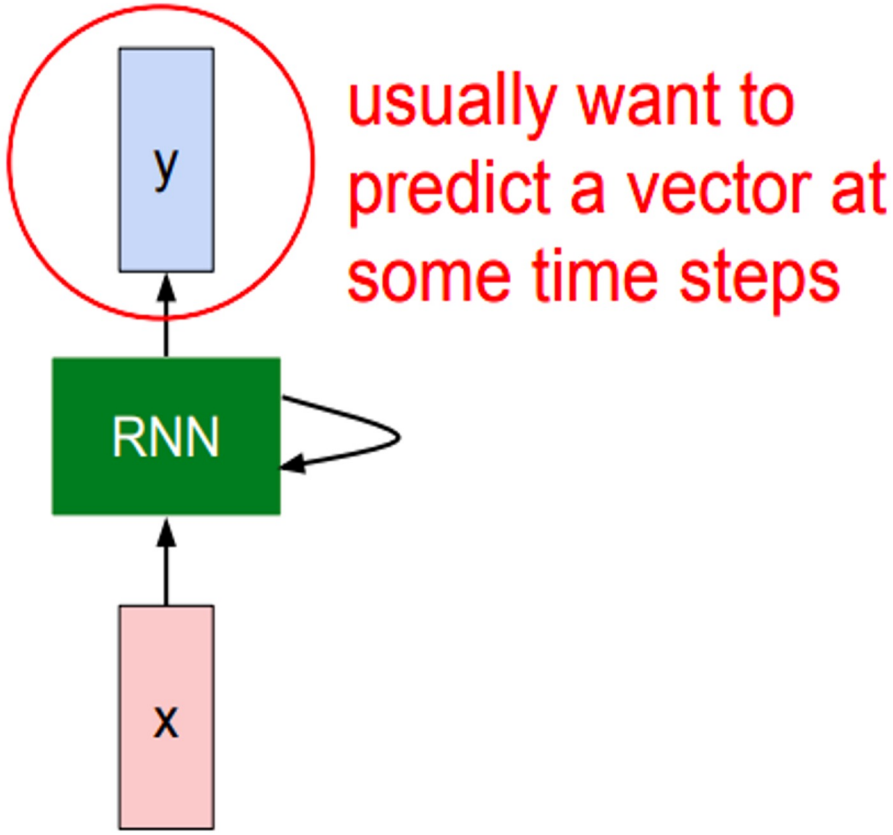


Credits: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture10.pdf

RNN architecture

Steps :

1. After tN steps produce output y



Credits: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture10.pdf

RNN architecture

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

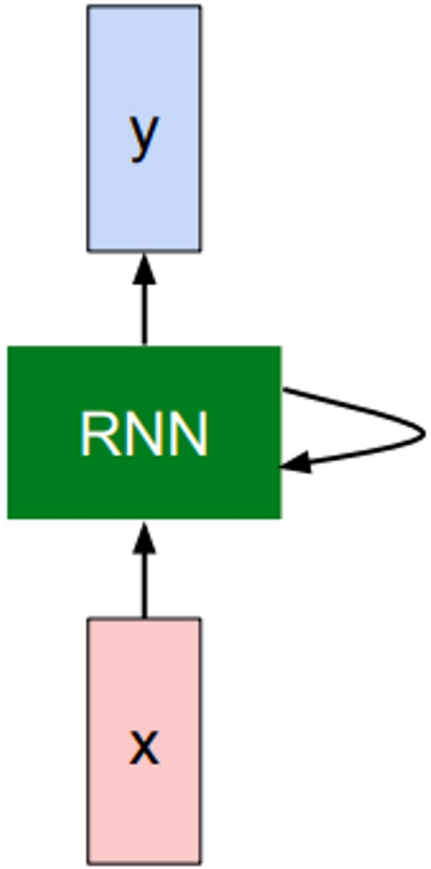
$$h_t = f_W(h_{t-1}, x_t)$$

new state

some function with parameters W

old state

input vector at some time step



Credits: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture10.pdf

RNN architecture

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

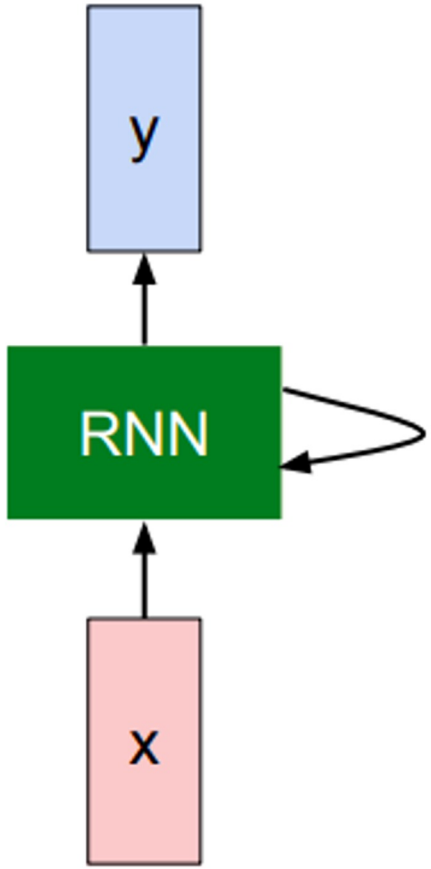
$$h_t = f_W(h_{t-1}, x_t)$$

new state

some function with parameters W

old state

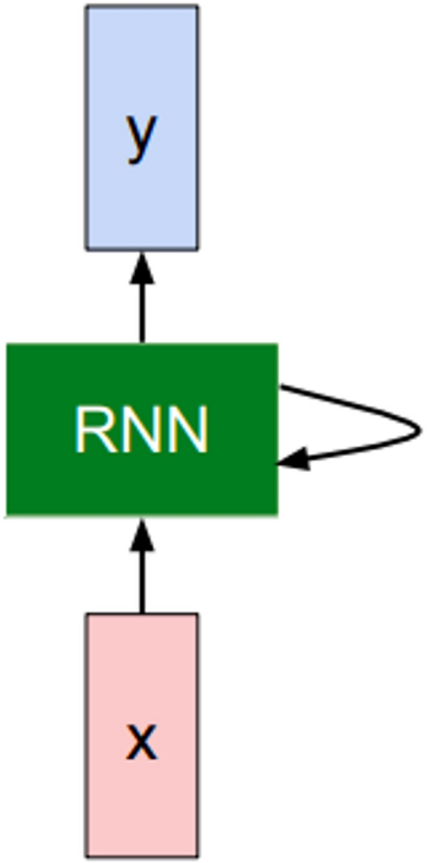
input vector at some time step



We use the same function f and parameters W for every timestep

Credits: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture10.pdf

RNN architecture : Vanilla RNN



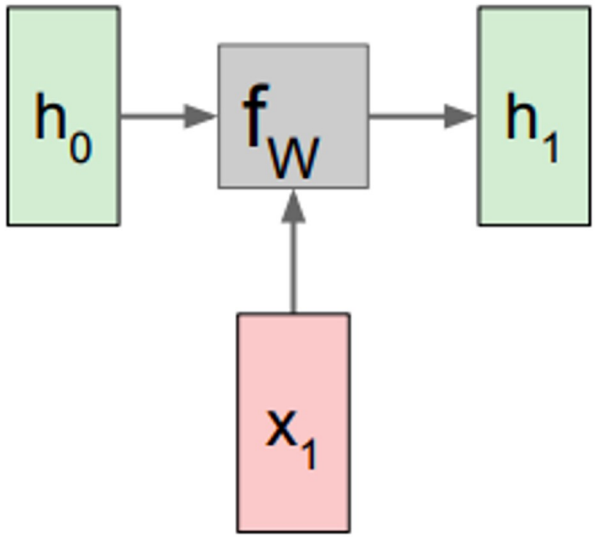
$$h_t = f_W(h_{t-1}, x_t)$$

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

$$y_t = W_{hy}h_t$$

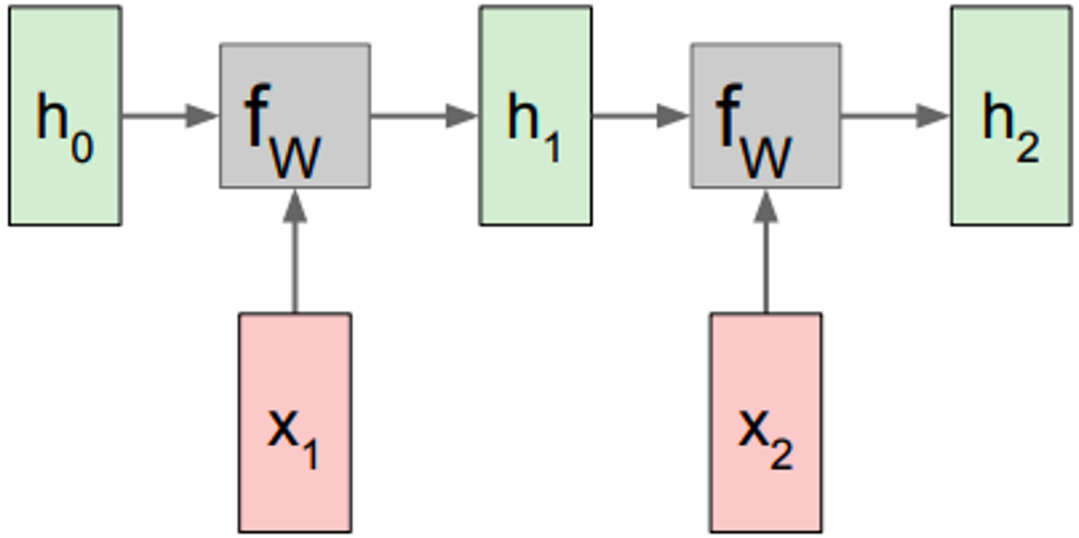
Credits: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture10.pdf

RNN architecture : Unrolling through time



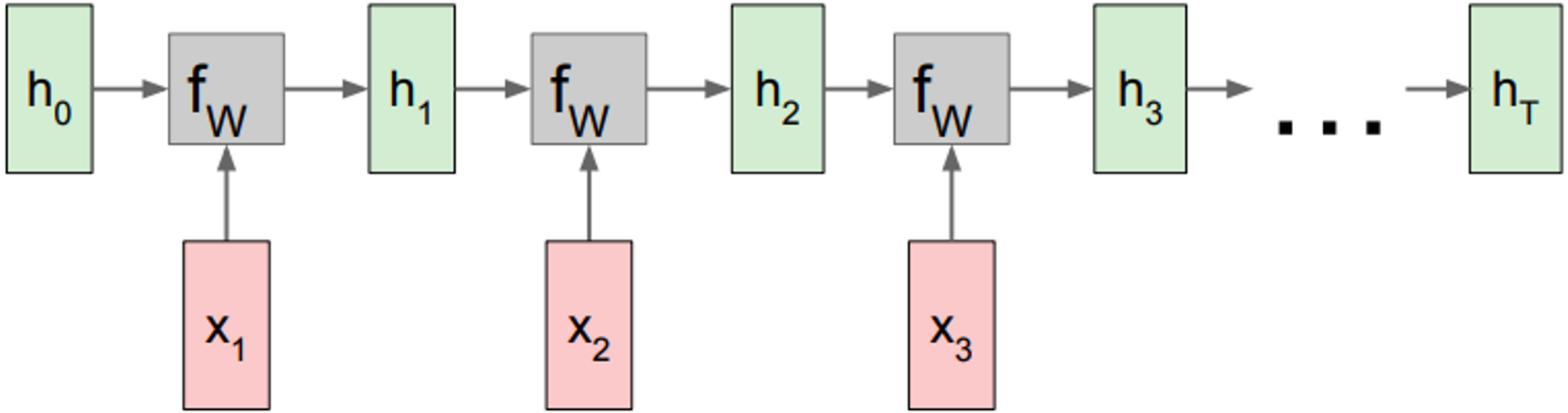
Credits: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture10.pdf

RNN architecture : Unrolling through time



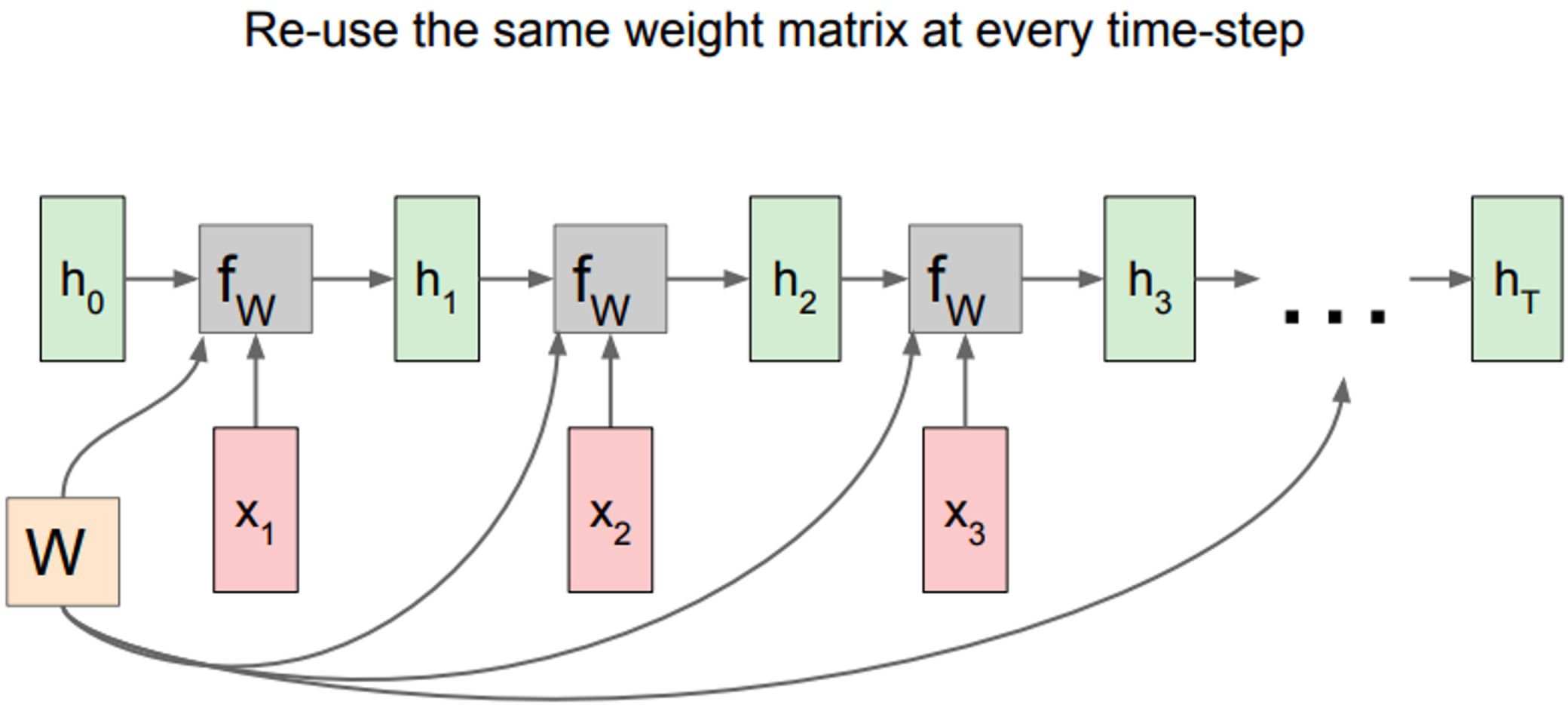
Credits: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture10.pdf

RNN architecture : Unrolling through time



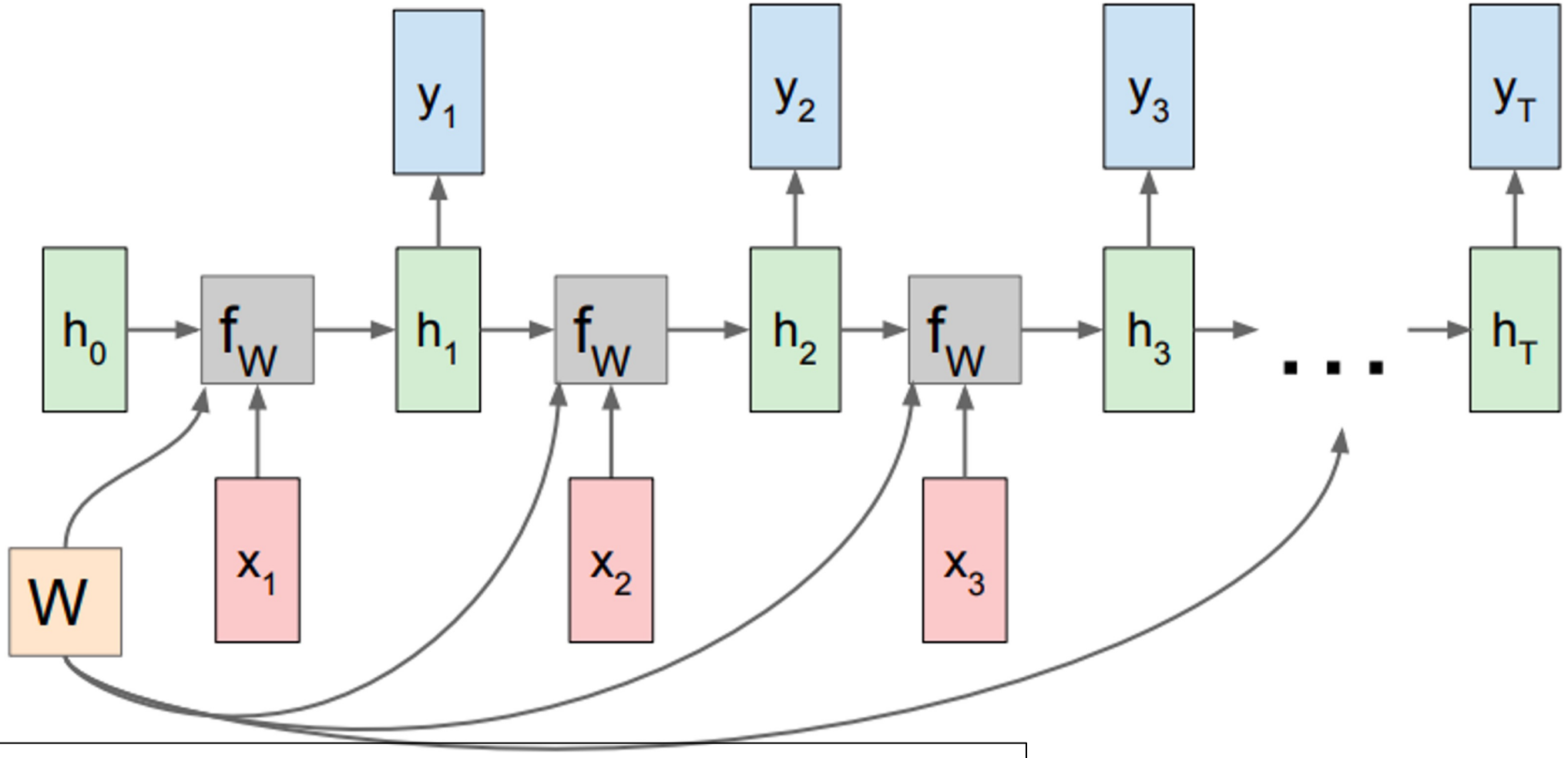
Credits: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture10.pdf

RNN architecture : Unrolling through time



Credits: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture10.pdf

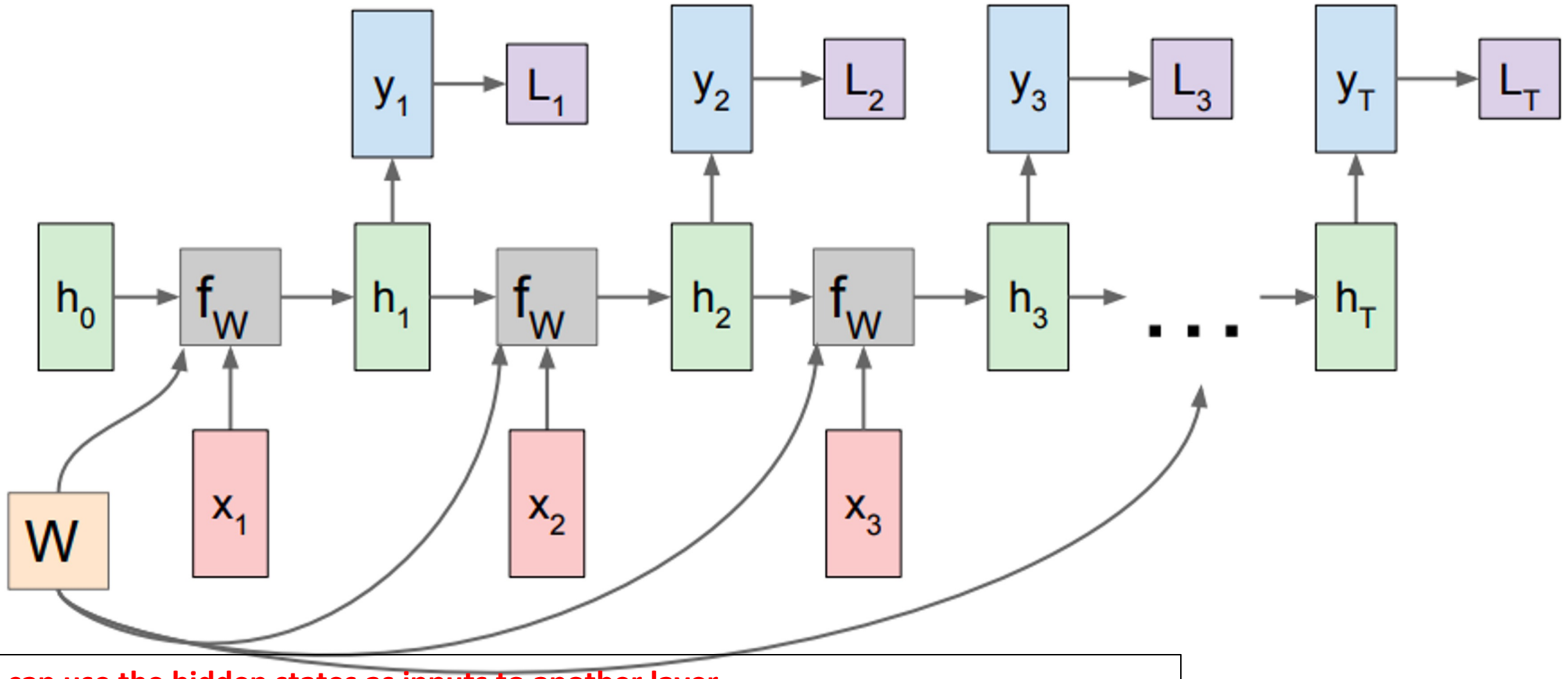
RNN architecture : Many to Many



We can use the hidden states as inputs to another layer

Credits: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture10.pdf

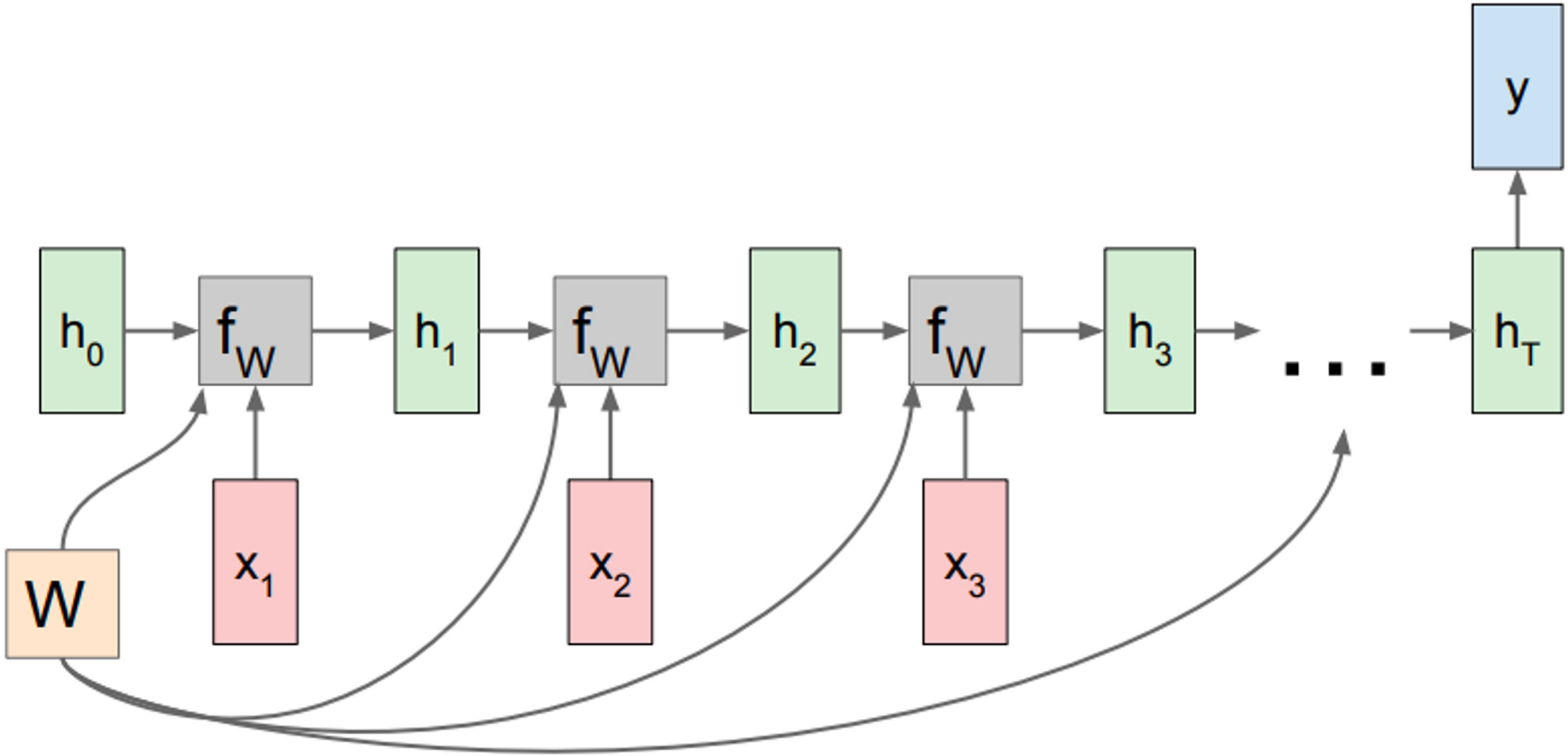
RNN architecture : Many to Many



We can use the hidden states as inputs to another layer by backpropagating losses

Credits: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture10.pdf

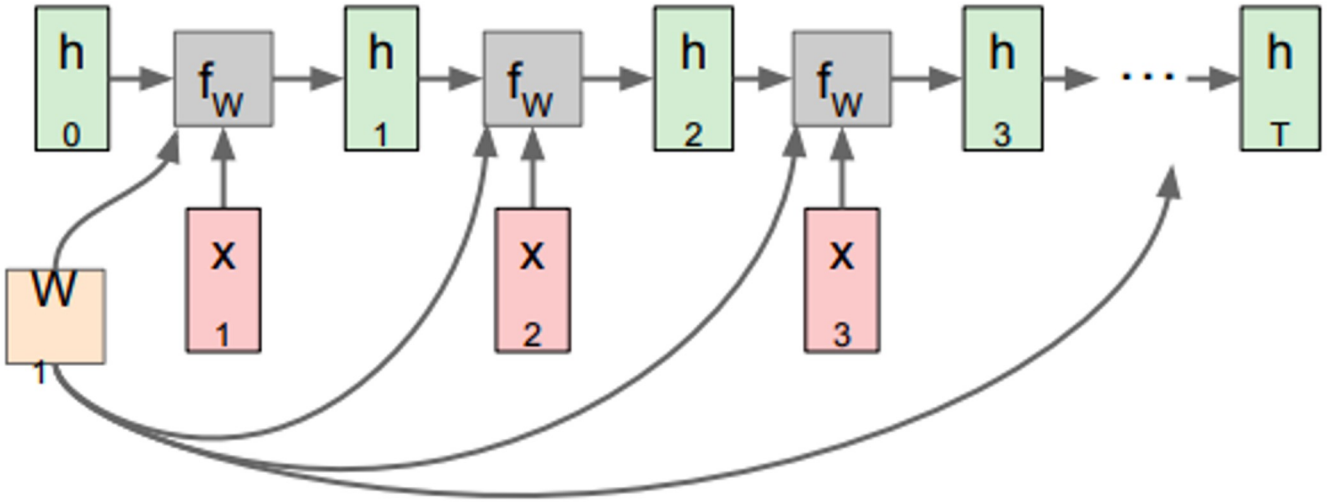
RNN architecture : Many to One



Credits: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture10.pdf

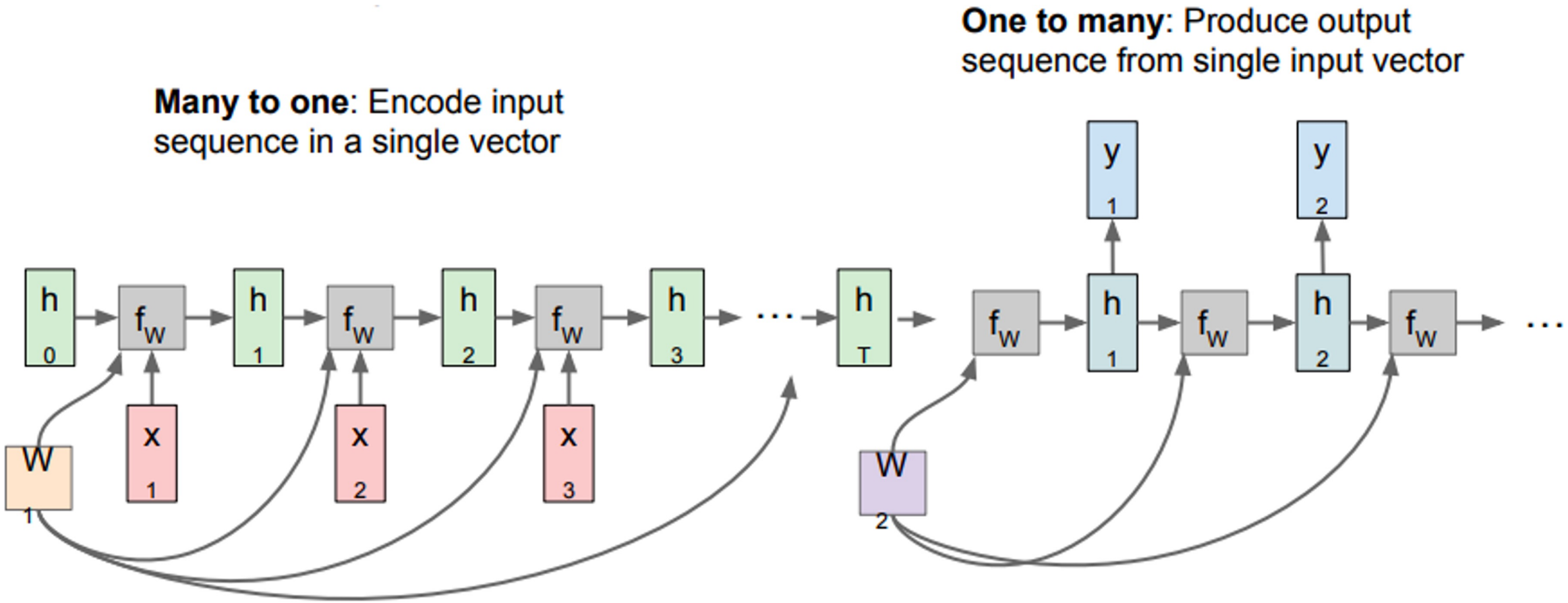
RNN architecture : Encoder decoder : Many to One + One to many

Many to one: Encode input sequence in a single vector



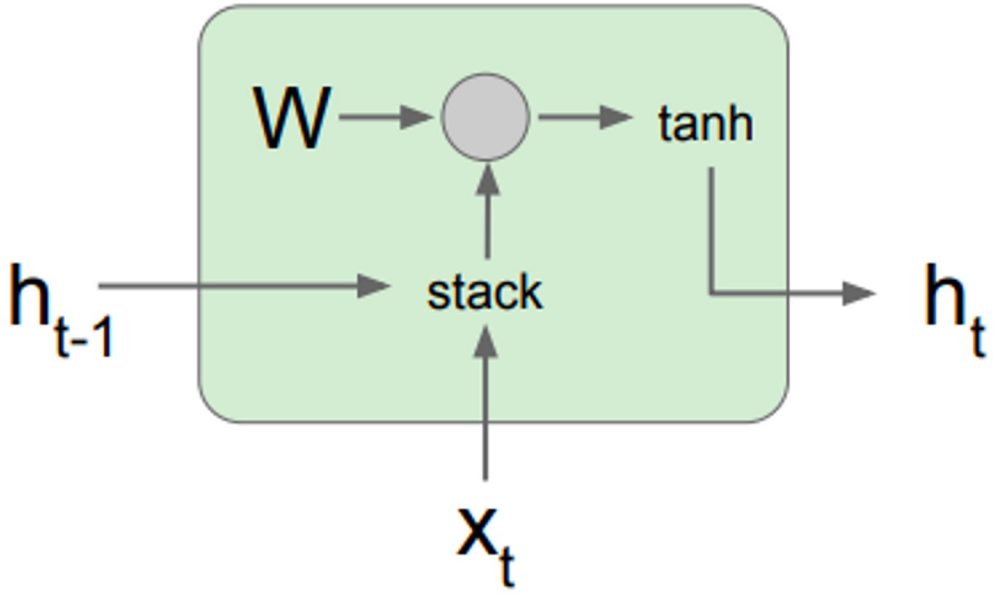
Credits: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture10.pdf

RNN architecture : Encoder decoder : Many to One + One to many



Credits: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture10.pdf

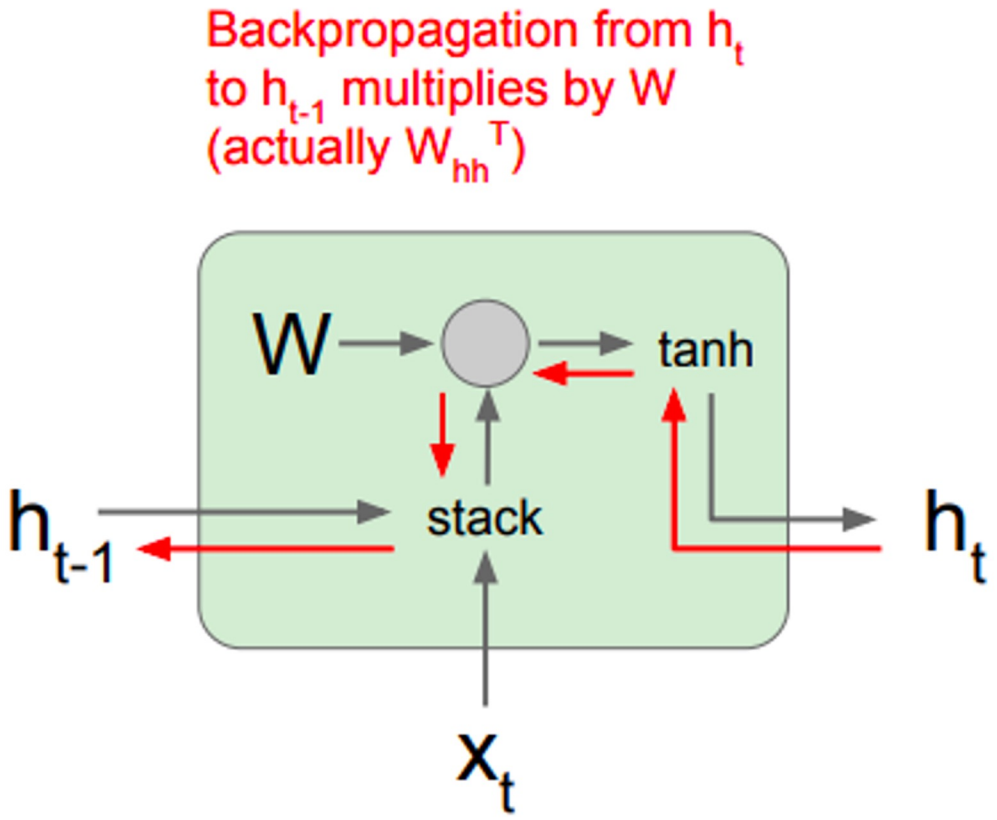
Gradient Flow in RNNs



$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$
$$= \tanh\left((W_{hh} \quad W_{hx})\begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)$$
$$= \tanh\left(W\begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)$$

Credits: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture10.pdf

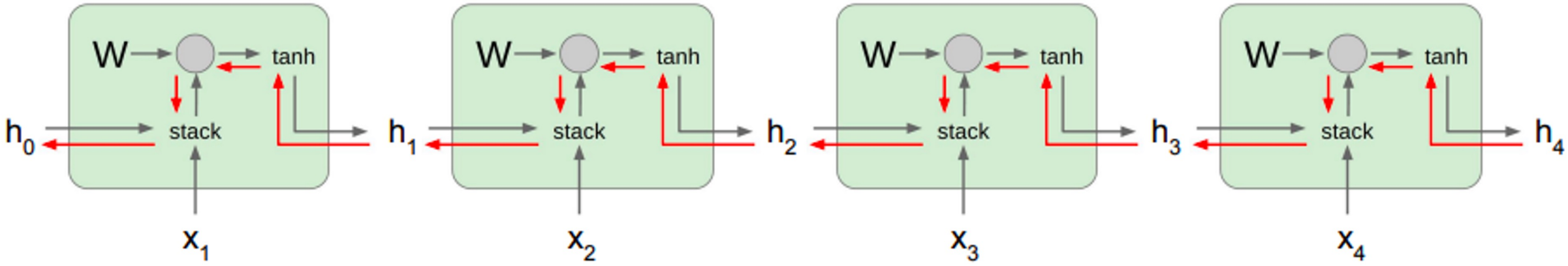
Gradient Flow in RNNs



$$
\begin{aligned}
h_t &= \tanh(W_{hh}h_{t-1} + W_{xh}x_t) \\
&= \tanh\left(\begin{pmatrix} W_{hh} & W_{hx} \end{pmatrix} \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} \right) \\
&= \tanh\left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} \right)
\end{aligned}
$$

Credits: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture10.pdf

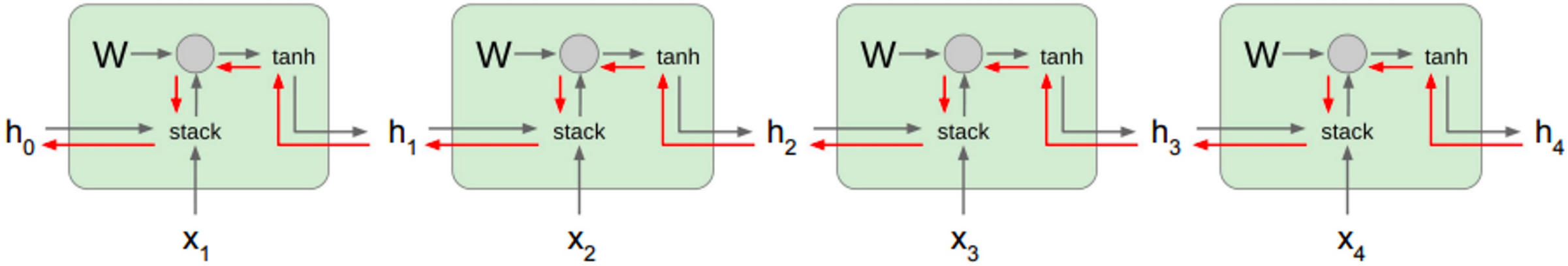
Gradient Flow in RNNs



Computing gradient of h_0 involves many factors of W (and repeated tanh)

Credits: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture10.pdf

Gradient Flow in RNNs



Computing gradient of h_0 involves many factors of W (and repeated tanh)

Largest singular value > 1:
Exploding gradients

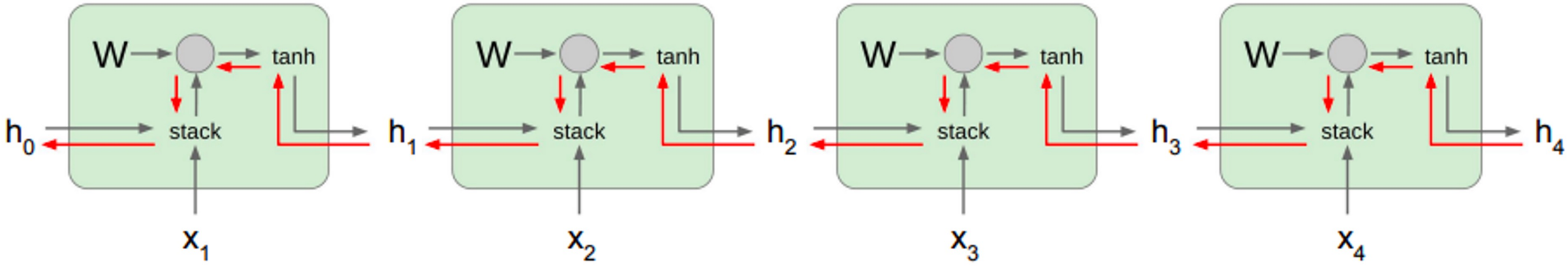
Scalar**N>>>1, scalar=2,

Largest singular value < 1:
Vanishing gradients

Scalar**N<<<1, scalar=0.2

Credits: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture10.pdf

Gradient Flow in RNNs



Computing gradient of h_0 involves many factors of W (and repeated tanh)

Largest singular value > 1:
Exploding gradients

Do gradient clipping : Scale the gradients

Largest singular value < 1:
Vanishing gradients

Change architecture, LSTMS!

Credits: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture10.pdf

LSTM

Vanilla RNN

$$h_t = \tanh\left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)$$

LSTM

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

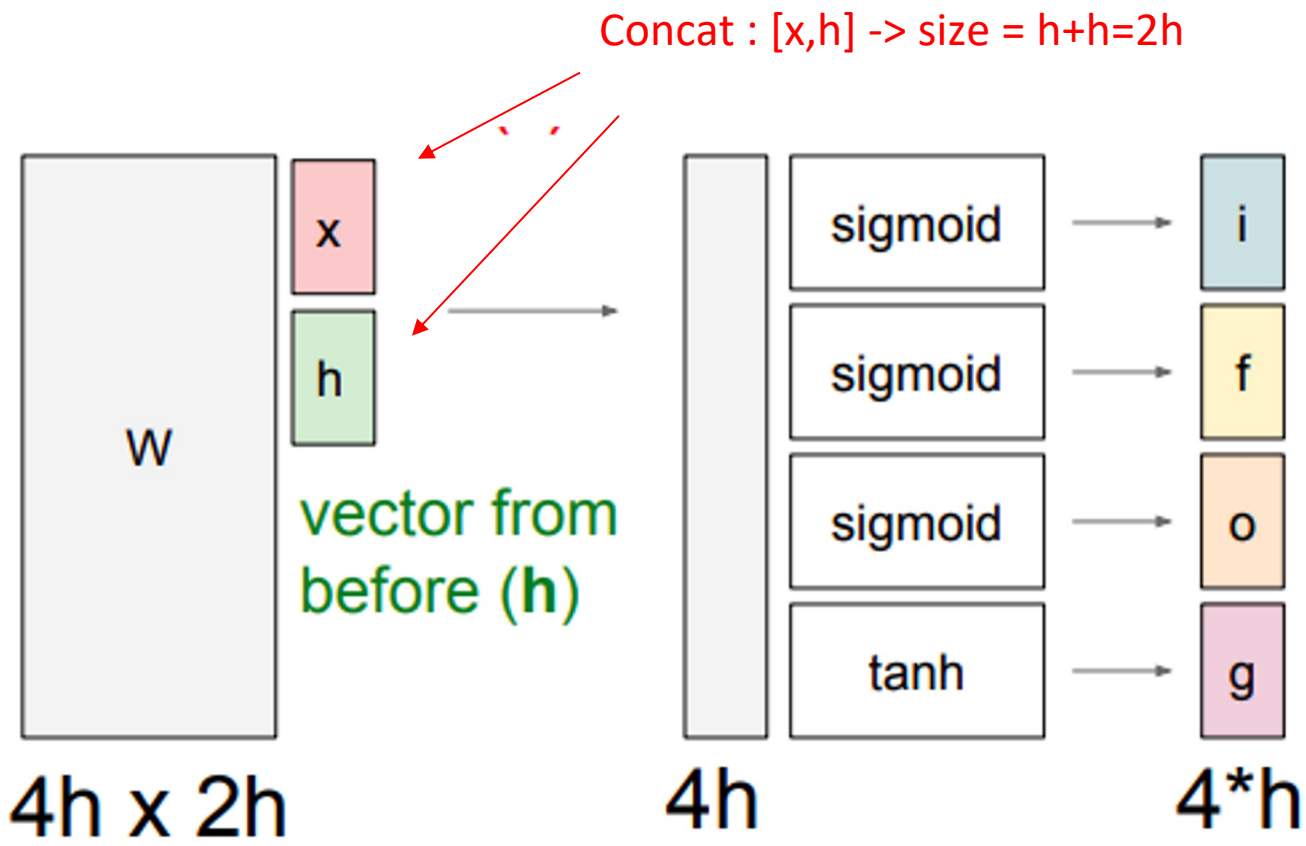
Cell state : remains inside forever

Hidden state : the same as RNNs

Element wise

Credits: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture10.pdf

LSTM



Input gate: whether to write to the cell

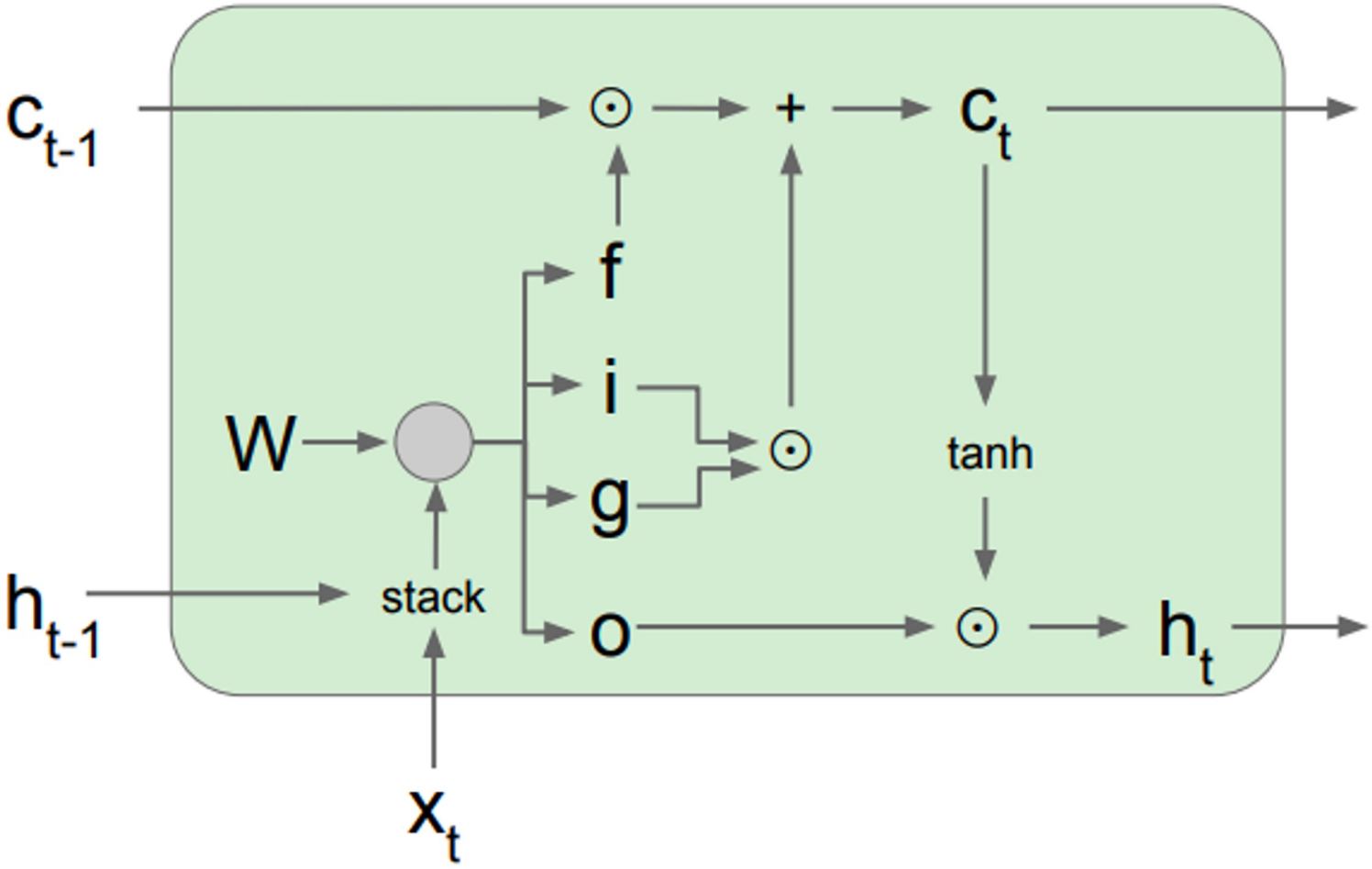
forget gate: whether to erase from the cell

output gate: how much to output from the cell

Gate gate: how much to write to the cell

Credits: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture10.pdf

LSTM



$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

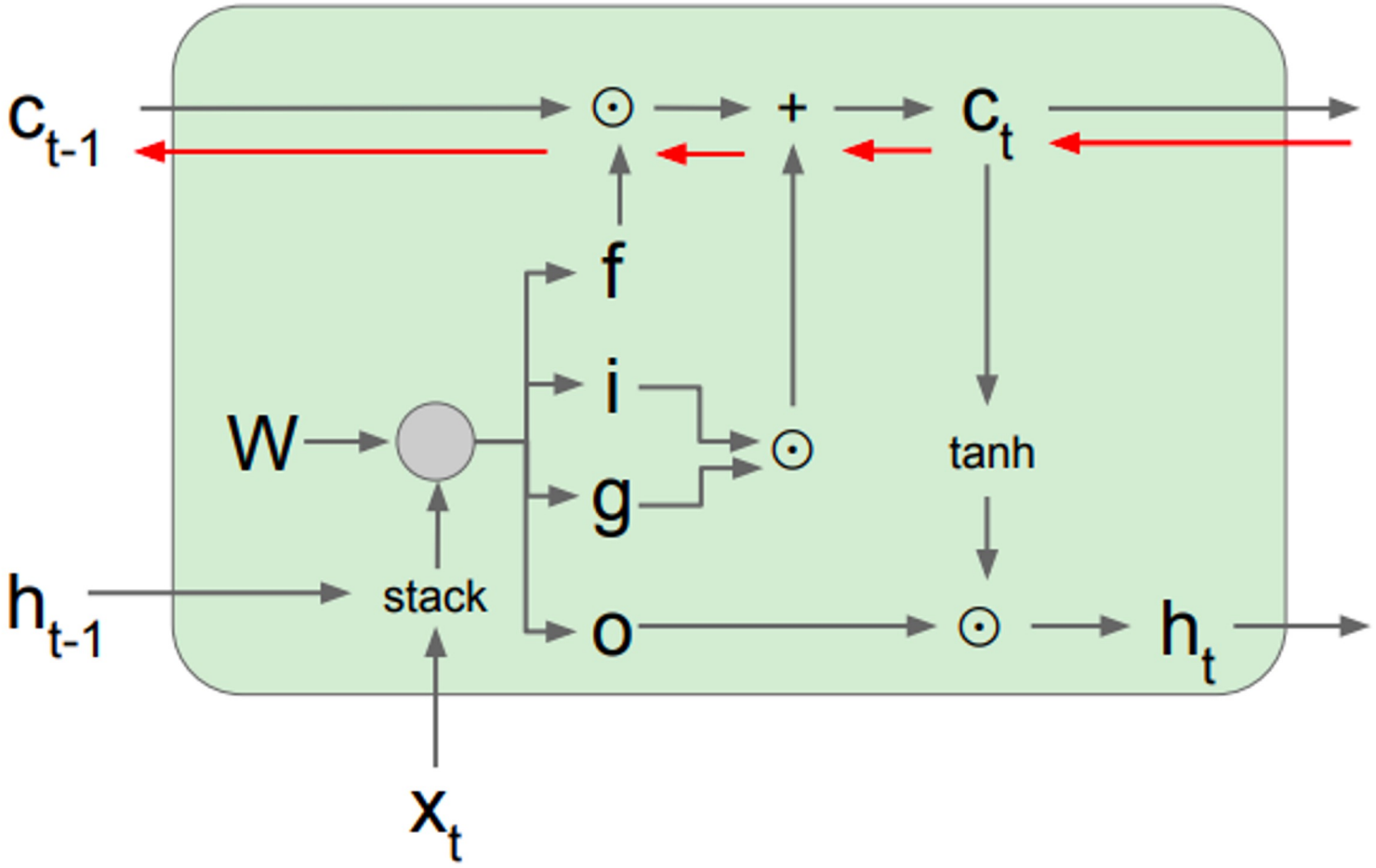
$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

Credits: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture10.pdf

LSTM

Local gradient for c(t-1) = f
Upstream grad = g
Total grad = g * f where * is element wise



Backpropagation from c_t to c_{t-1} only elementwise multiplication by f, no matrix multiply by W

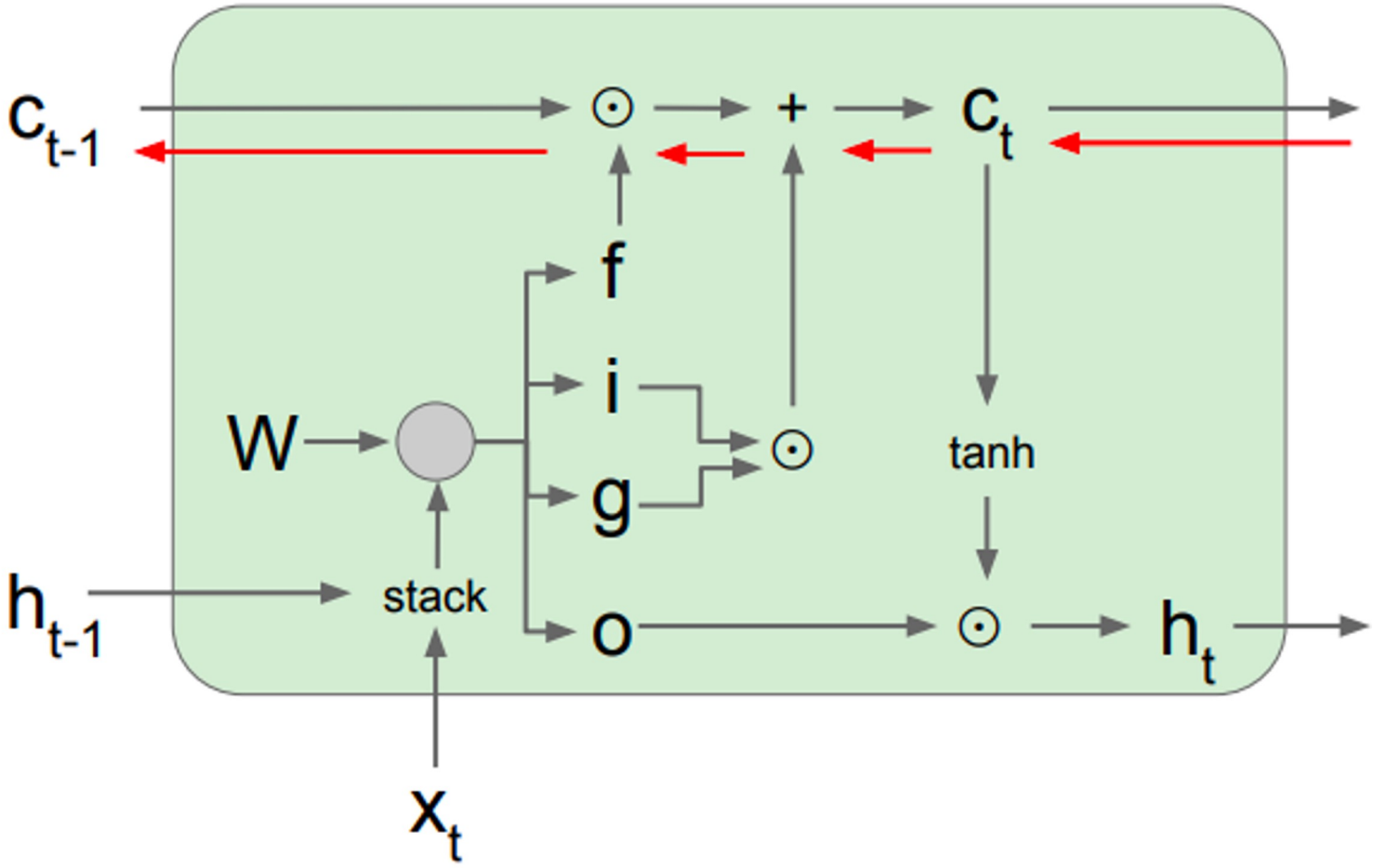
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

Credits: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture10.pdf

LSTM

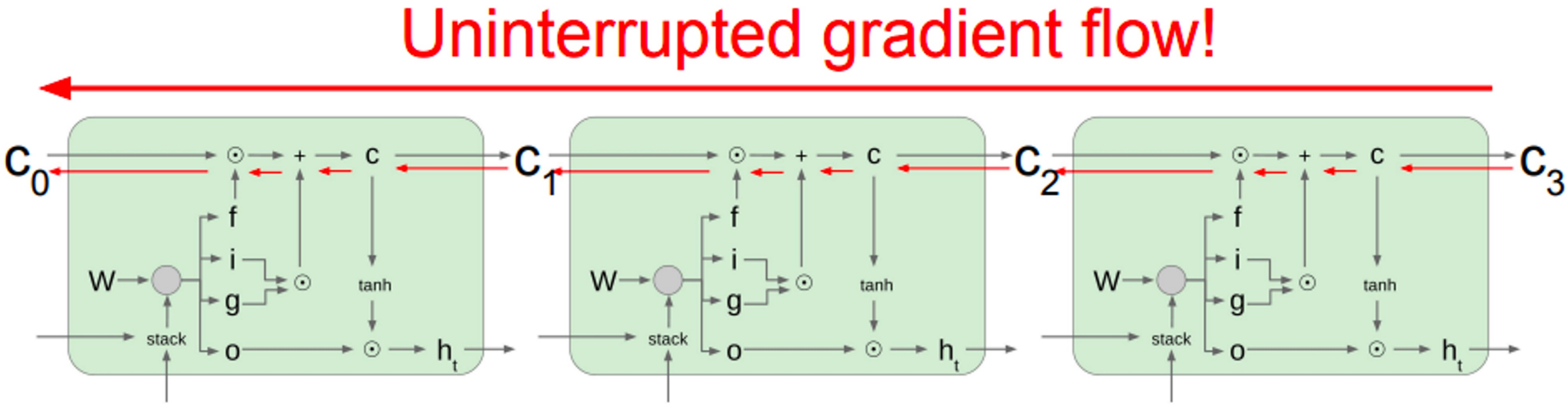


Gradients behave properly because of :

1. Elementwise multiplication by forget gate
2. Forget gate keeps changing at every time step
3. Forget gate comes out of a sigmoid so every element is between 0 and 1

Credits: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture10.pdf

LSTM



Credits: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture10.pdf

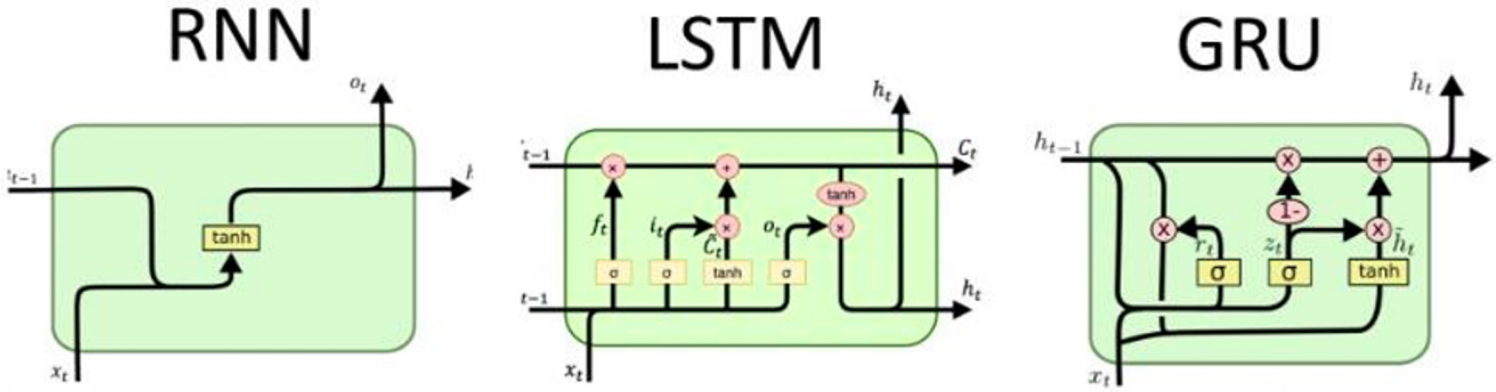
GRU

LSTM Disadvantages :
Slower and heavier than RNN

Credits: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture10.pdf

GRU

Idea: combine long-term and short-term updates in single channel



More or less behaves like LSTM but lighter parameters, so faster to train

Credits: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture10.pdf

GRU

GRU [*Learning phrase representations using rnn encoder-decoder for statistical machine translation*, Cho et al. 2014]

$$r_t = \sigma(W_{xr}x_t + W_{hr}h_{t-1} + b_r)$$
$$z_t = \sigma(W_{xz}x_t + W_{hz}h_{t-1} + b_z)$$
$$\tilde{h}_t = \tanh(W_{xh}x_t + W_{hh}(r_t \odot h_{t-1}) + b_h)$$
$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot \tilde{h}_t$$

[*LSTM: A Search Space Odyssey*, Greff et al., 2015]

More or less behaves like LSTM but lighter parameters, so faster to train

Credits: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture10.pdf

Next Steps

Implement simple RNN models in tensorflow keras

Credits: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture10.pdf