Machine Learning: Algorithms and Applications

Philip O. Ogunbona

Advanced Multimedia Research Lab
University of Wollongong

Artificial Neural Networks and Deep Learning: An Introduction (IV)

Parts of these slides are from Fei Fei Li.

Outline

Types of time series data

Time series (sequence) data (Chollet 2021)

A time series can be any data obtained via measurements at regular intervals

- daily price of a stock
- hourly electricity consumption
- weekly sales of a store
- seismic activity
- evolution of fish populations in a river
- human activity patterns

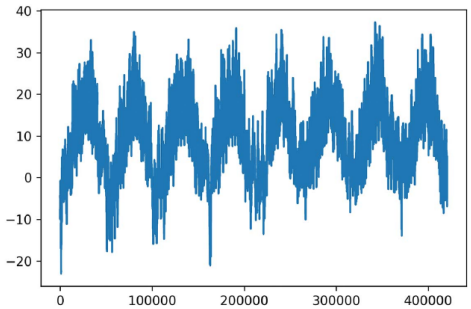


Figure 1: Temperature: full temporal range of the dataset; every 10 min (℃) (Chollet 2021)

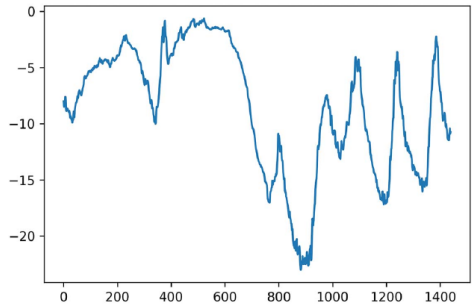


Figure 2: Temperature: the first 10 days of the dataset; every 10 min (℃) (Chollet 2021)

More characteristic of time series (sequence) data

- Each data point: A sequence of vectors $x(t)$, for $1 \leq t \leq \tau$
- Batch data: many sequences with different lengthsτ
- Label: can be a scalar, a vector, or even a sequence

Example:

- Sentiment analysis
- machine translation

More characteristics of time series (sequence) data

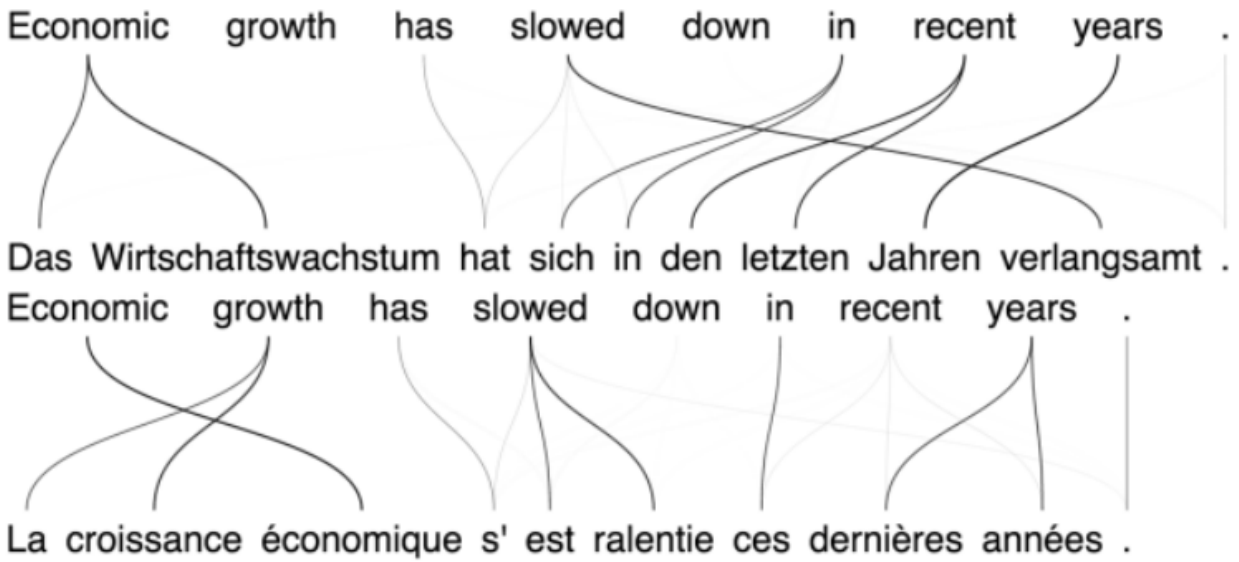


Figure 3: Machine translation (Figure from: `devblogs.nvidia.com`)

More characteristics of time series (sequence data)

- Data point: two dimensional sequences like images
- Label: different type of sequences like text sentences

Example:

- image captioning

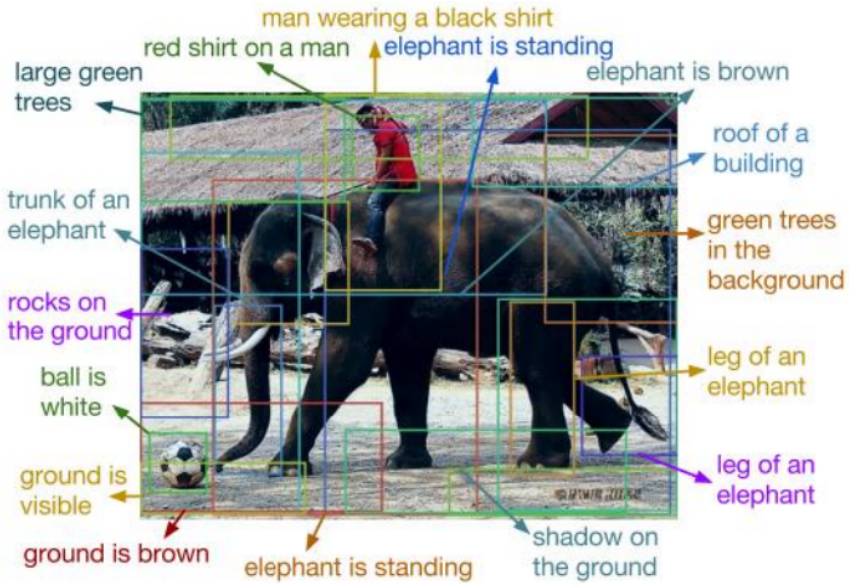


Figure 4: Image captioning (Figure from the paper “DenseCap: Fully Convolutional Localization Networks for Dense Captioning”, by Justin Johnson, Andrej Karpathy, Li Fei-Fei)

Recurrent neural network

- Biological intelligence processes information incrementally while maintaining an internal model of what is being processed based on past information and constantly updated as new information arrives.
- A recurrent neural network (RNN) adopts the similar principle, but extremely simplified version.
- Sequences are processed by iterating through the sequence elements and maintaining a state that contains information relative to what it has seen so far.
- RNN is a type of neural network that has an internal loop.

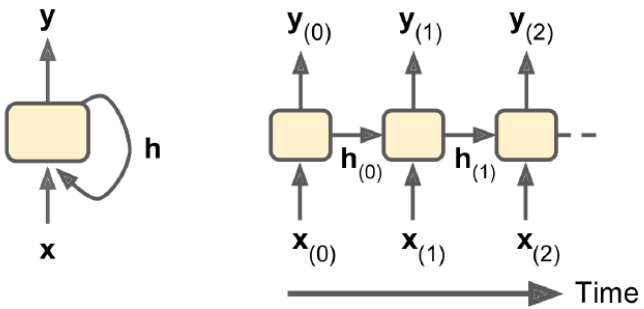


Figure 5: Simple RNN cell and unrolled version showing hidden state h

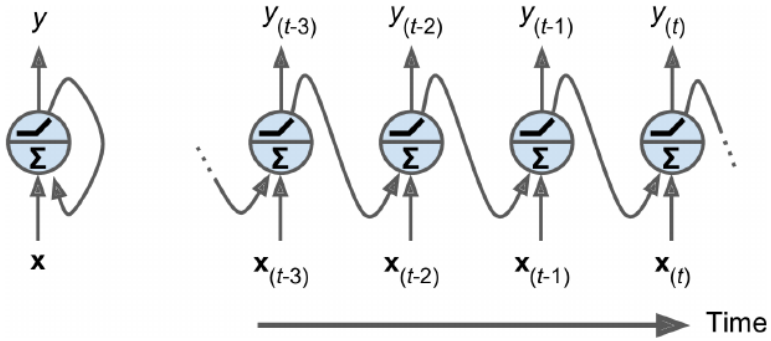


Figure 6: Simple RNN and Unrolled version

Recurrent neural network

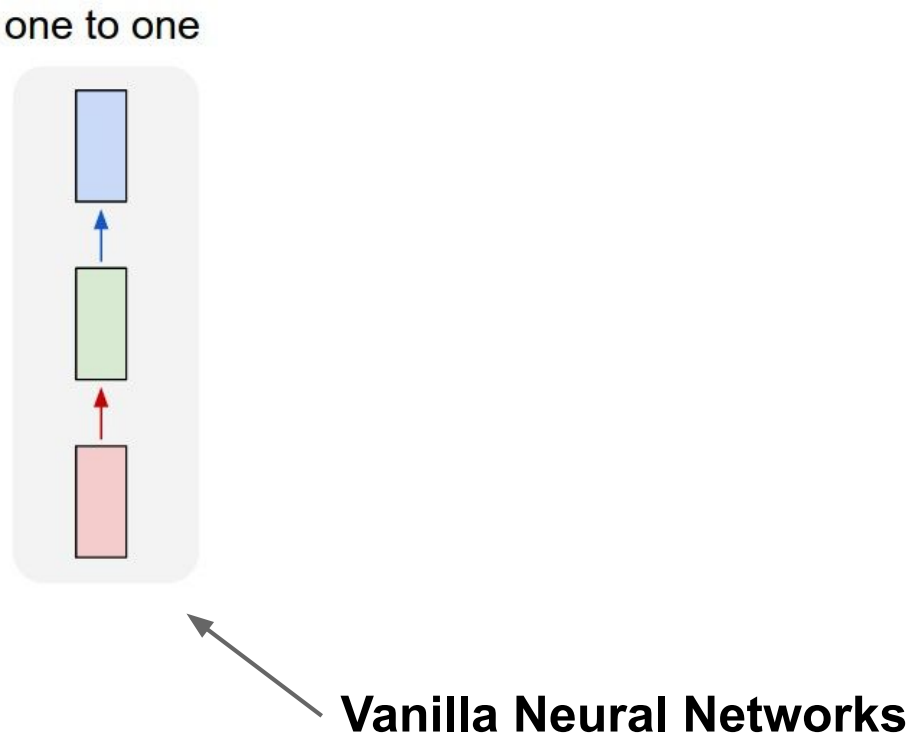
simple keras/tensorflow RNN

```
        model = keras.models.Sequential([
keras.layers.SimpleRNN(1, input_shape=[None, 1])
])
```

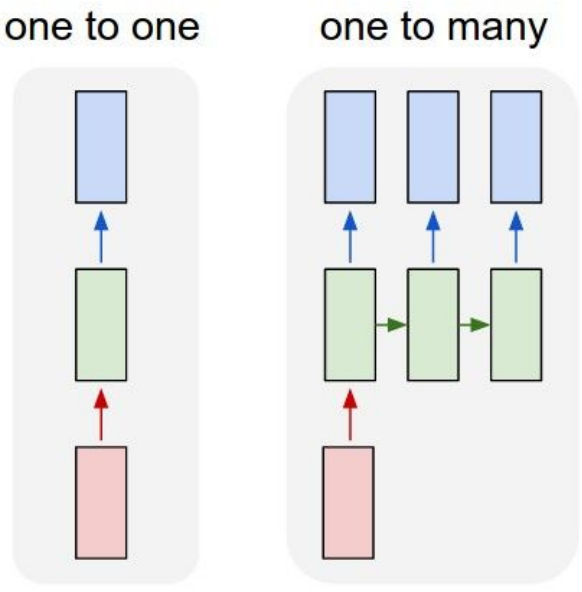
layers of RNN in keras/tensorflow

```
model = keras.models.Sequential([
keras.layers.SimpleRNN(20, return_sequences=True, input_shape=[None, 1]),
keras.layers.SimpleRNN(20, return_sequences=True),
keras.layers.SimpleRNN(1)
])
```

“Vanilla” Neural Network

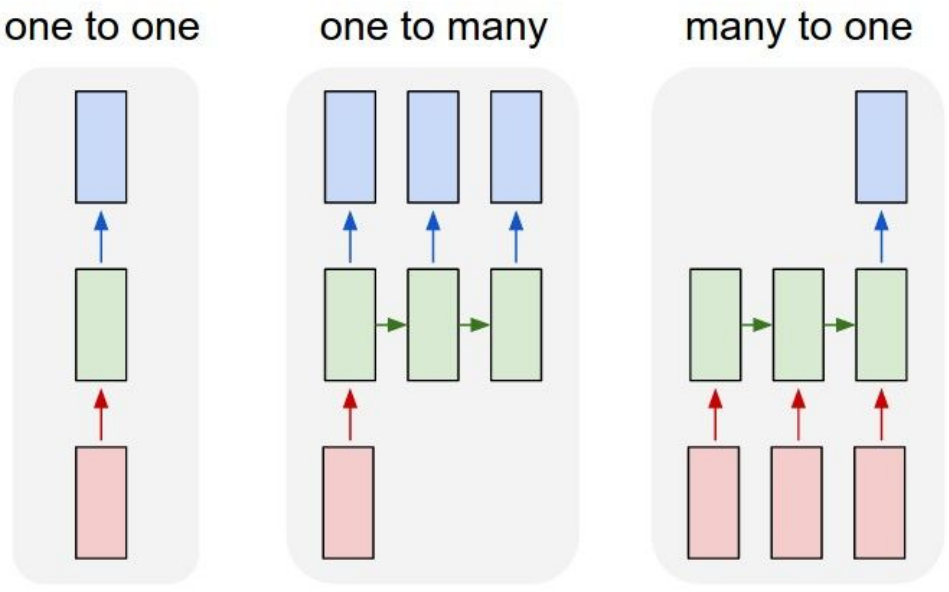


Recurrent Neural Networks: Process Sequences



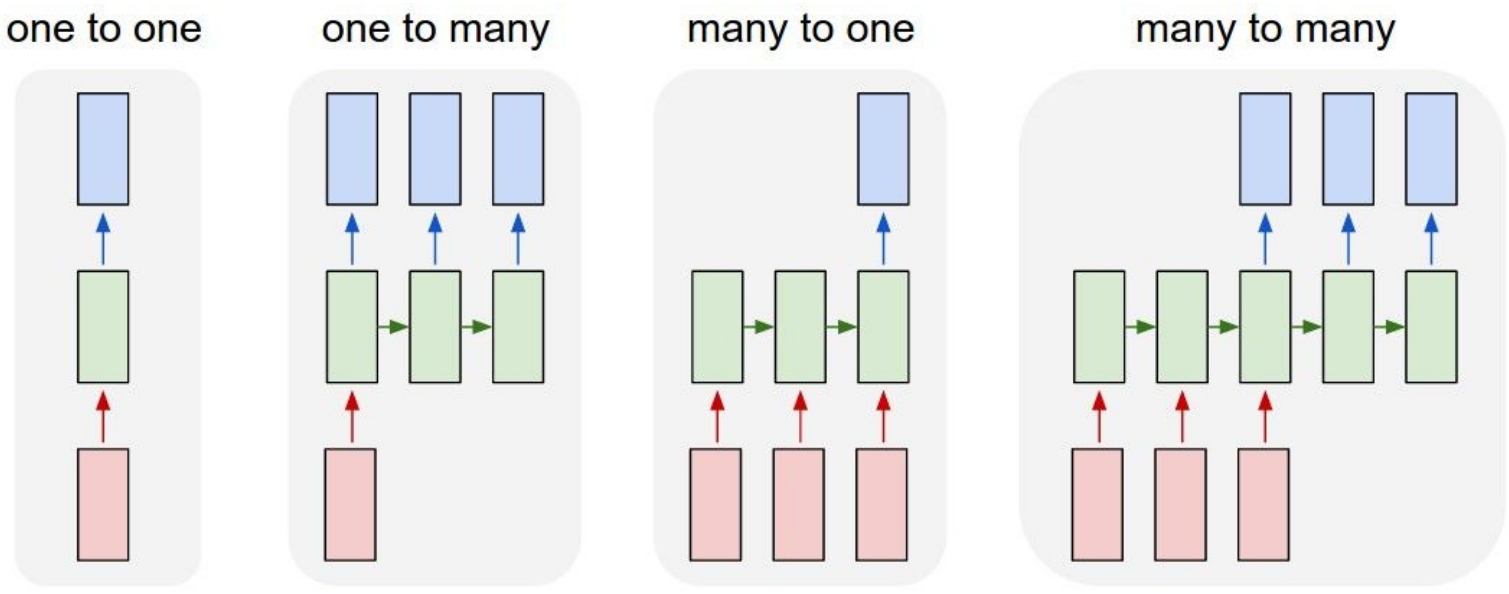
e.g. **Image Captioning**
image -> sequence of words

Recurrent Neural Networks: Process Sequences



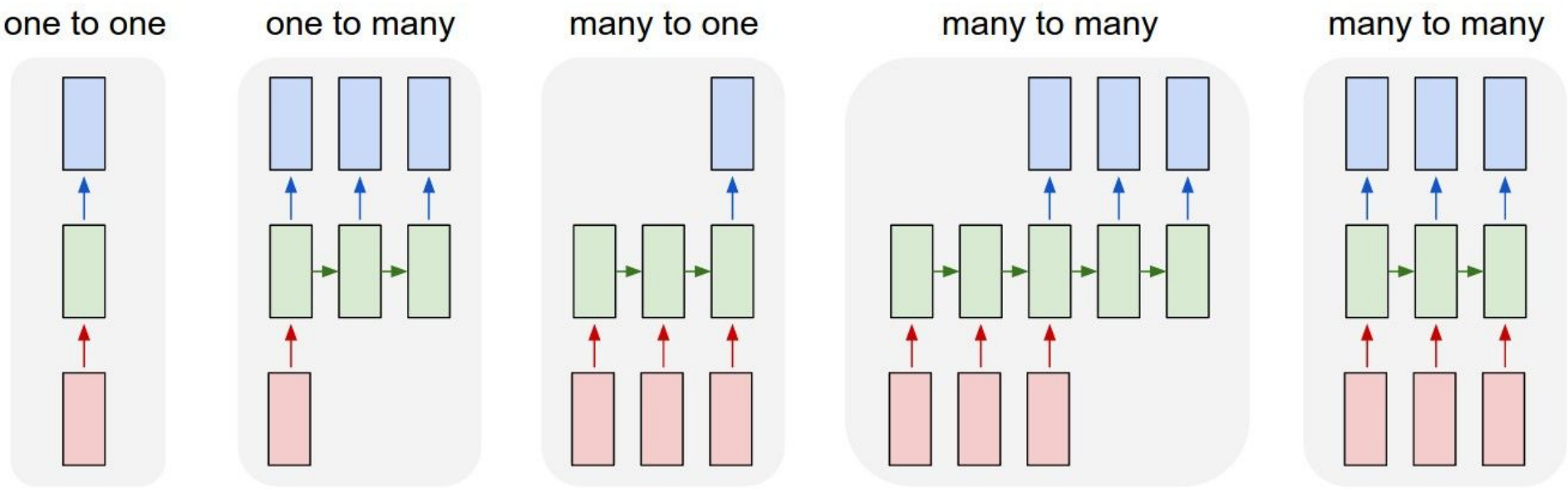
e.g. **action prediction**
sequence of video frames -> action class

Recurrent Neural Networks: Process Sequences



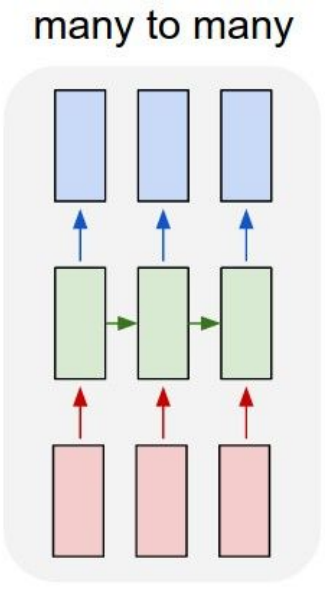
E.g. **Video Captioning**
Sequence of video frames -> caption

Recurrent Neural Networks: Process Sequences

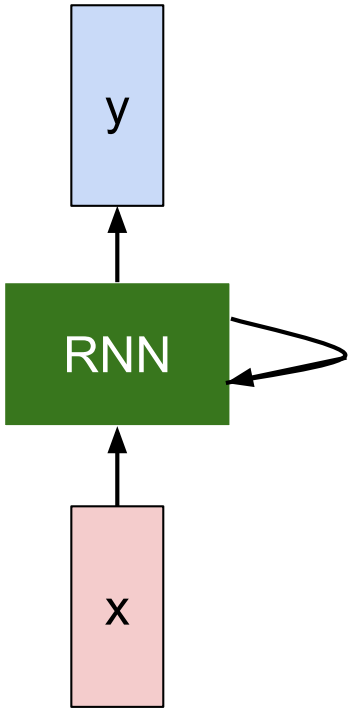


e.g. **Video classification on frame level**

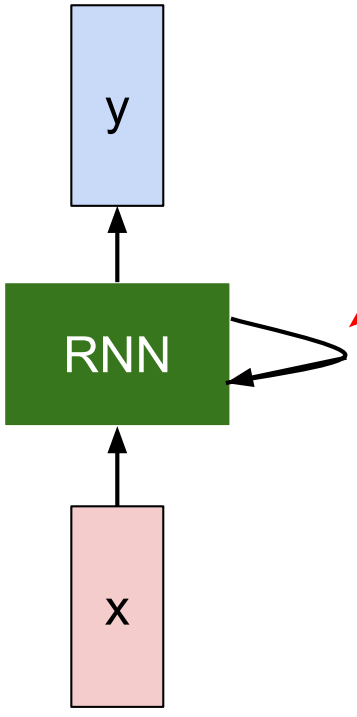
Let's start with a task that takes a variable input and produces an output at every step



Recurrent Neural Network

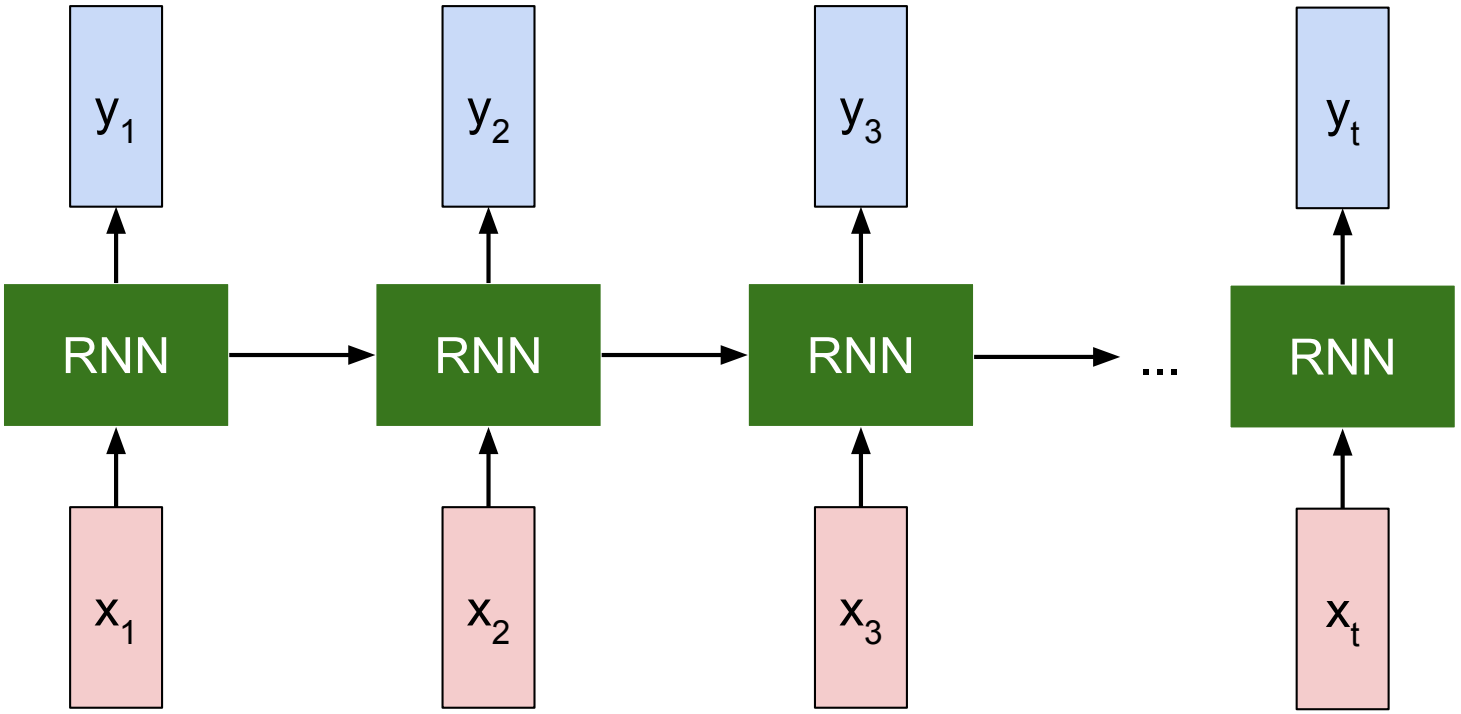


Recurrent Neural Network



Key idea: RNNs have an "internal state" that is updated as a sequence is processed

Unrolled RNN



RNN hidden state update

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

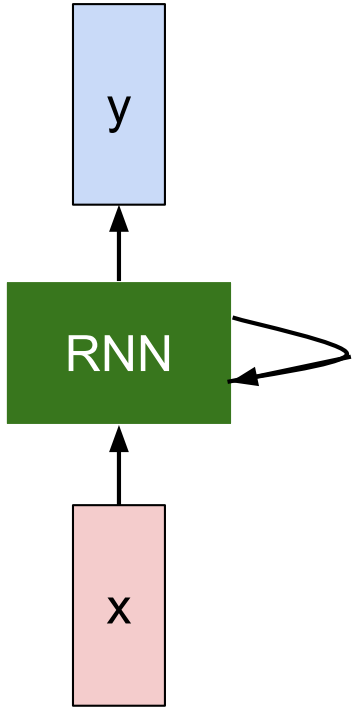
$$h_t = f_W(h_{t-1}, x_t)$$

new state

some function with parameters W

old state

input vector at some time step



RNN output generation

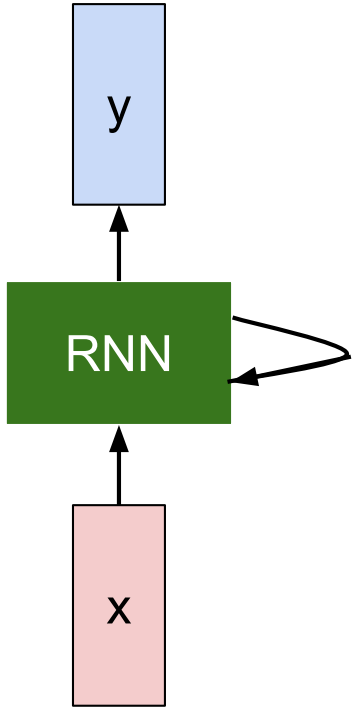
We can process a sequence of vectors **x** by applying a **recurrence formula** at every time step:

$$y_t = f_{W_{hy}}(h_t)$$

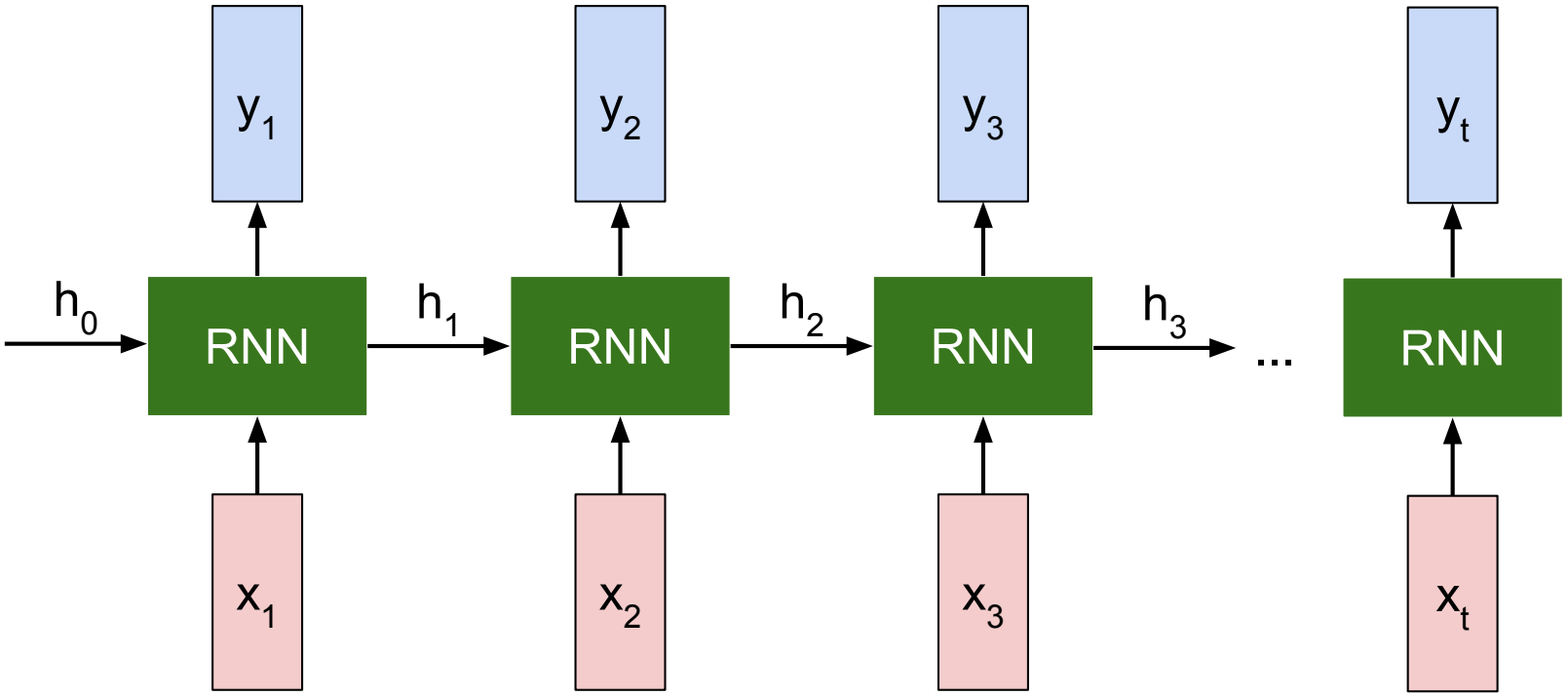
output

new state

another function with parameters W_o



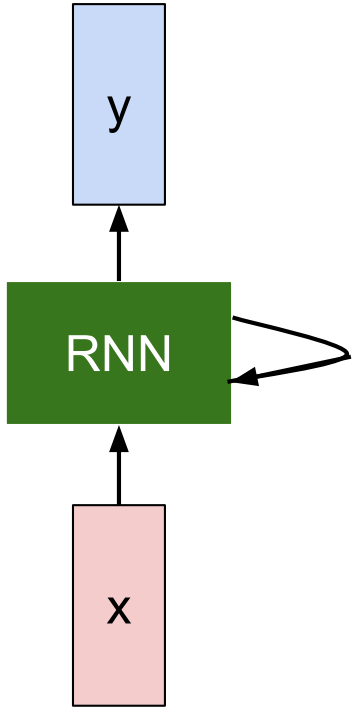
Recurrent Neural Network



Recurrent Neural Network

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

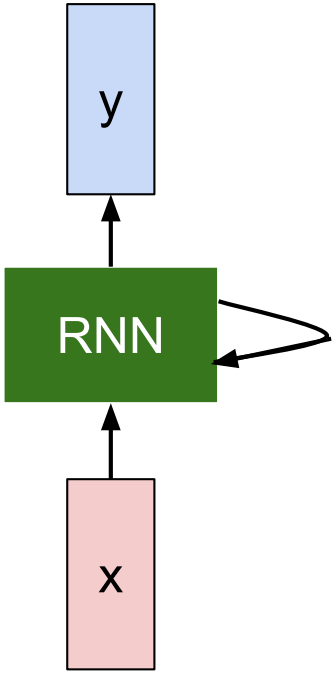
$$h_t = f_W(h_{t-1}, x_t)$$



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

(Simple) Recurrent Neural Network

The state consists of a single *"hidden"* vector **h**:



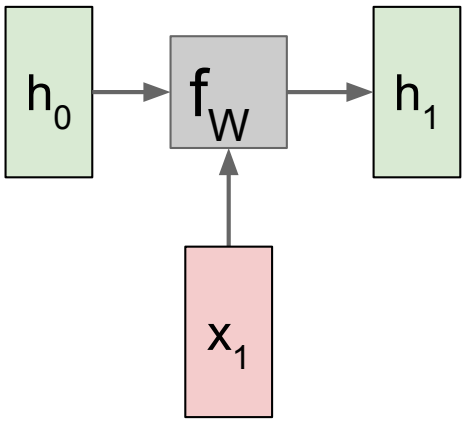
$$h_t = f_W(h_{t-1}, x_t)$$

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

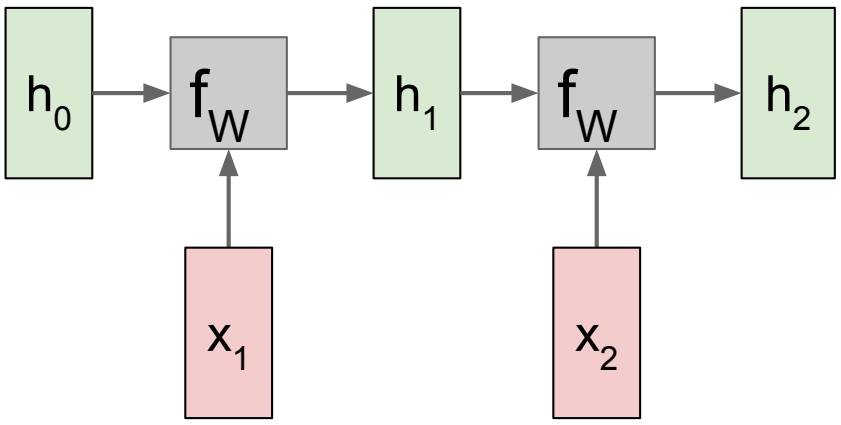
$$y_t = W_{hy}h_t$$

Sometimes called a "Vanilla RNN" or an "Elman RNN" after Prof. Jeffrey Elman

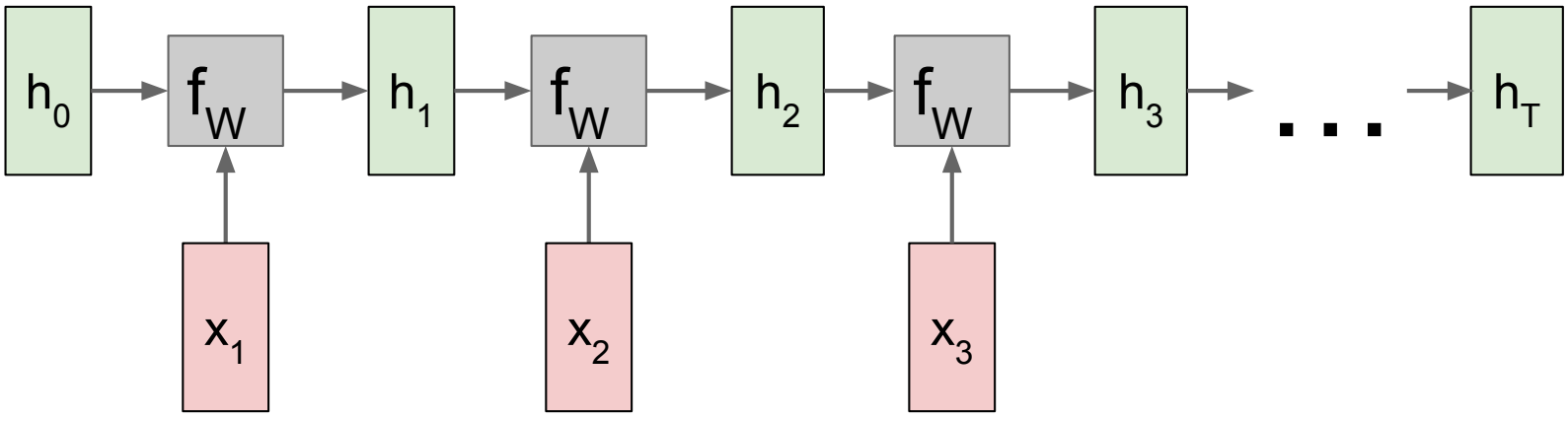
RNN: Computational Graph



RNN: Computational Graph

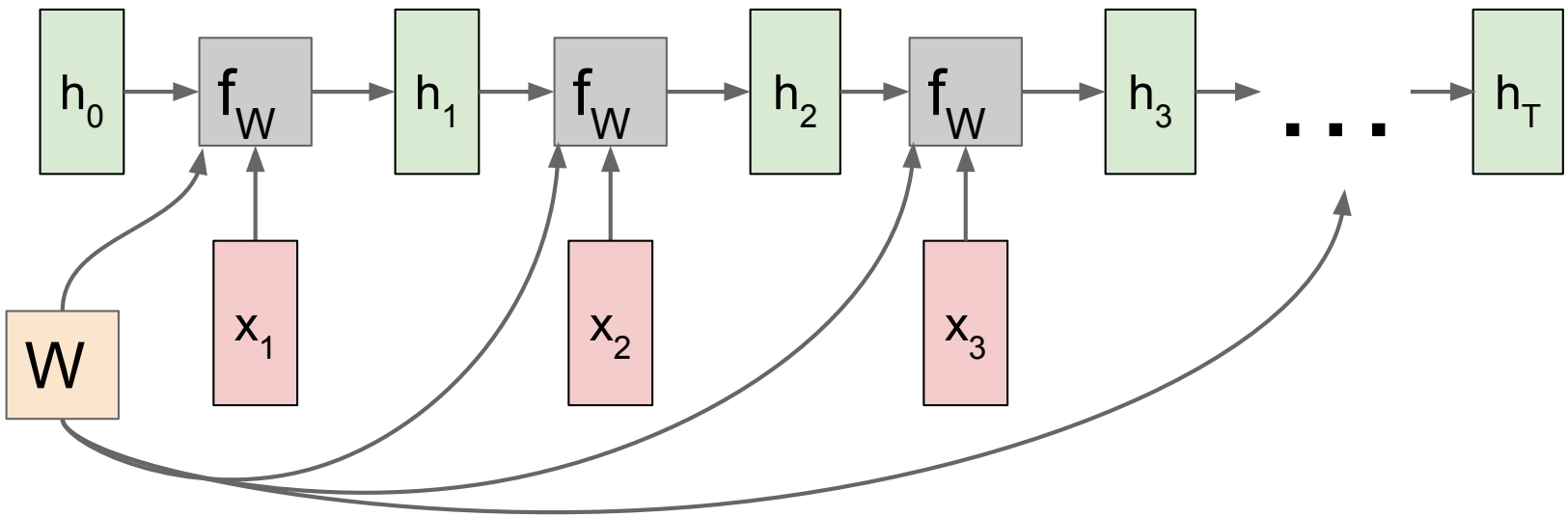


RNN: Computational Graph

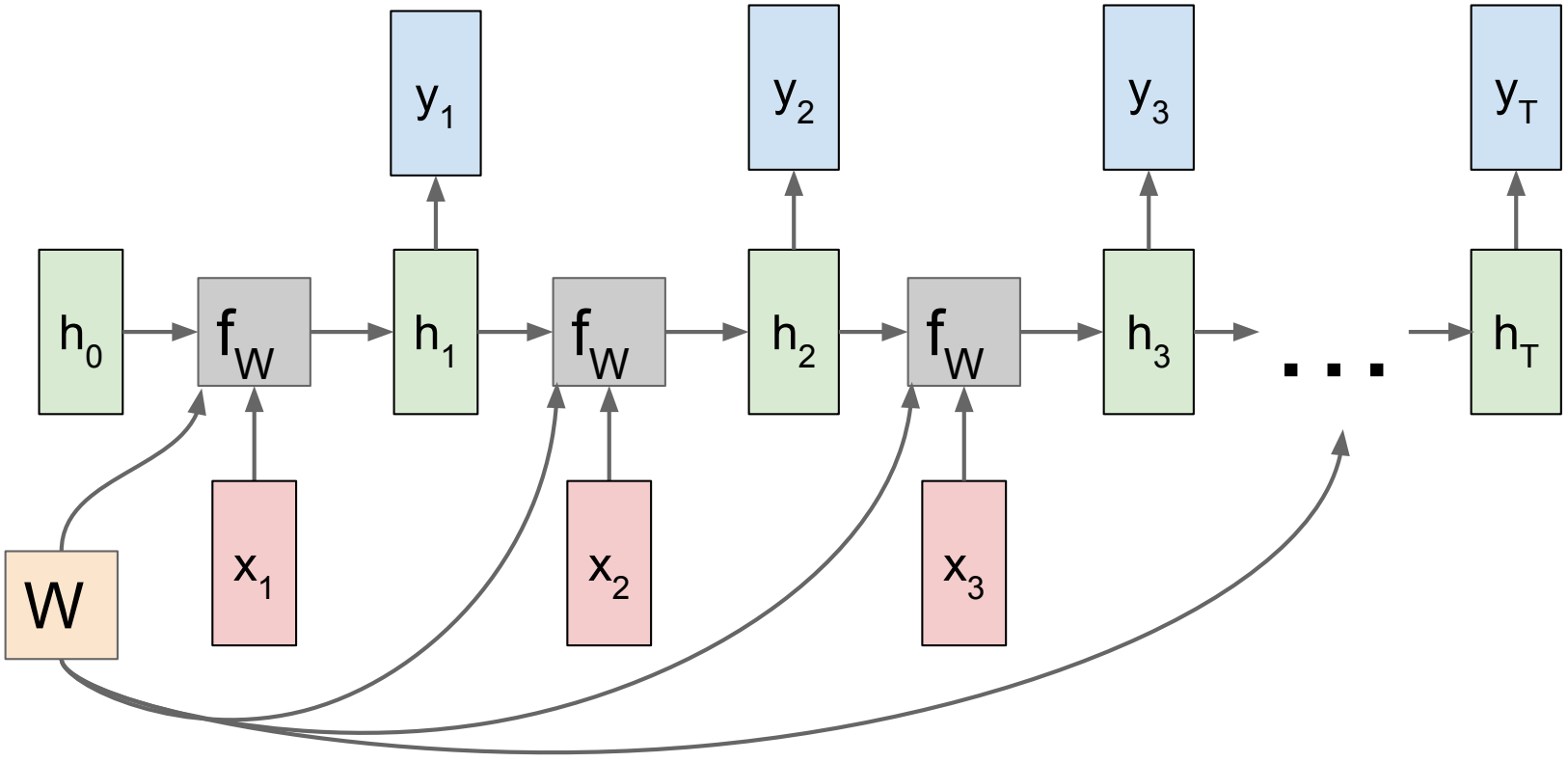


RNN: Computational Graph

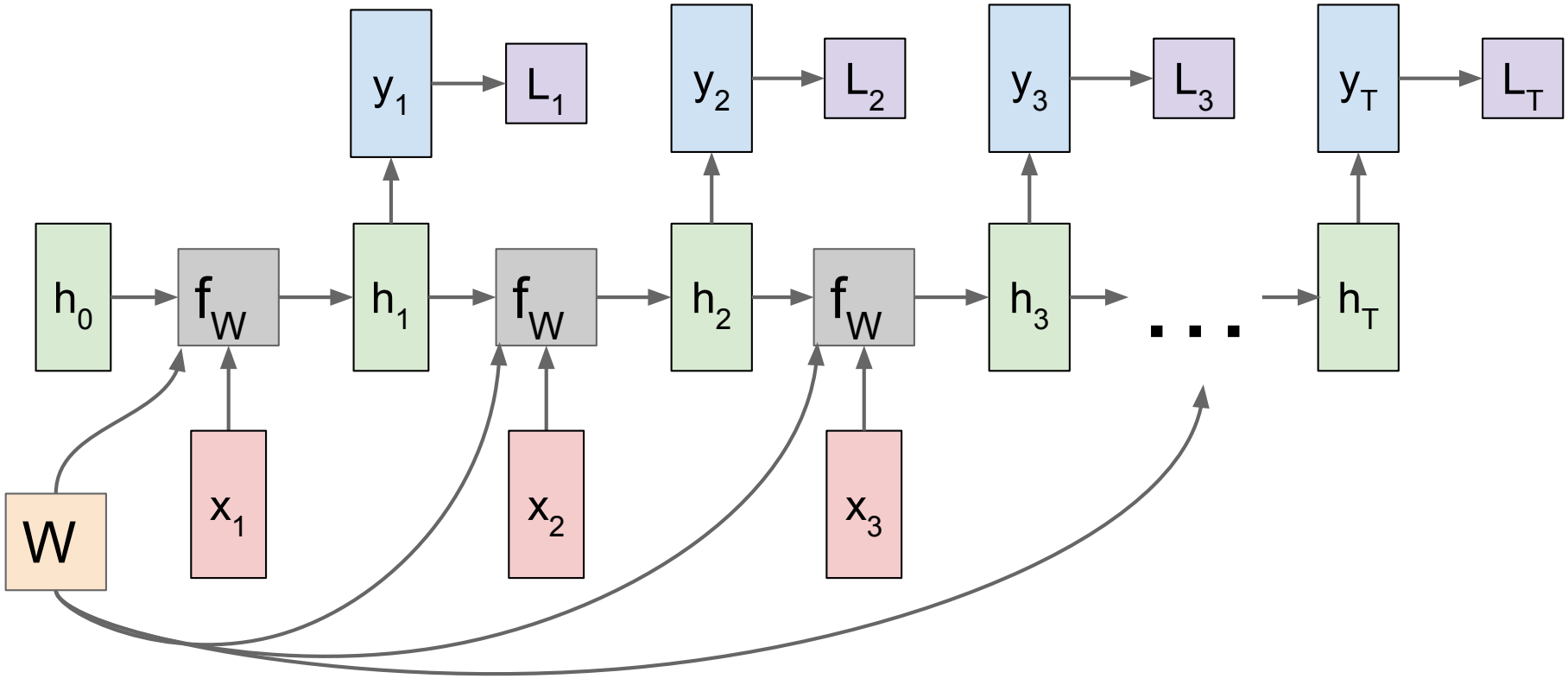
Re-use the same weight matrix at every time-step



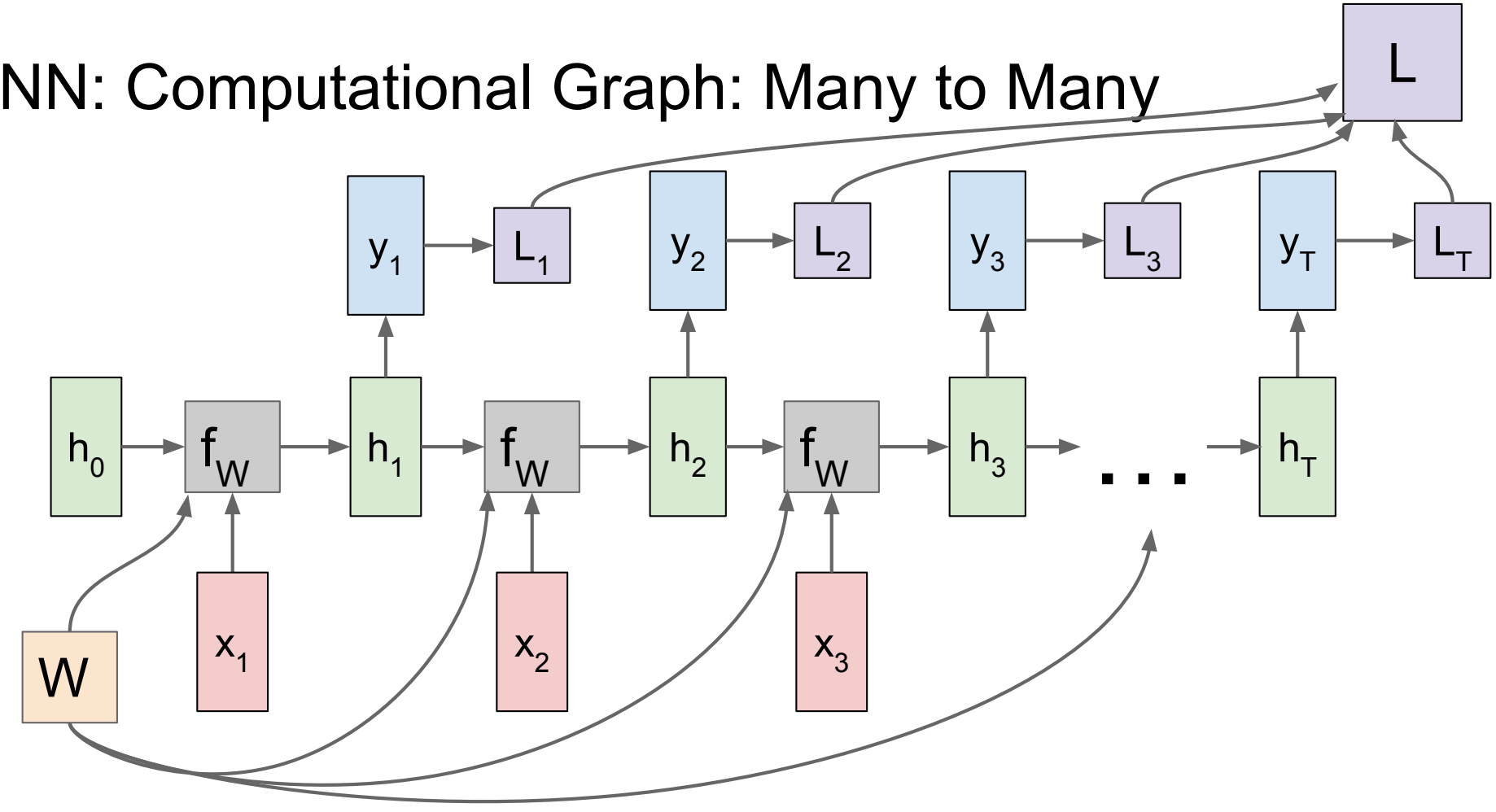
RNN: Computational Graph: Many to Many



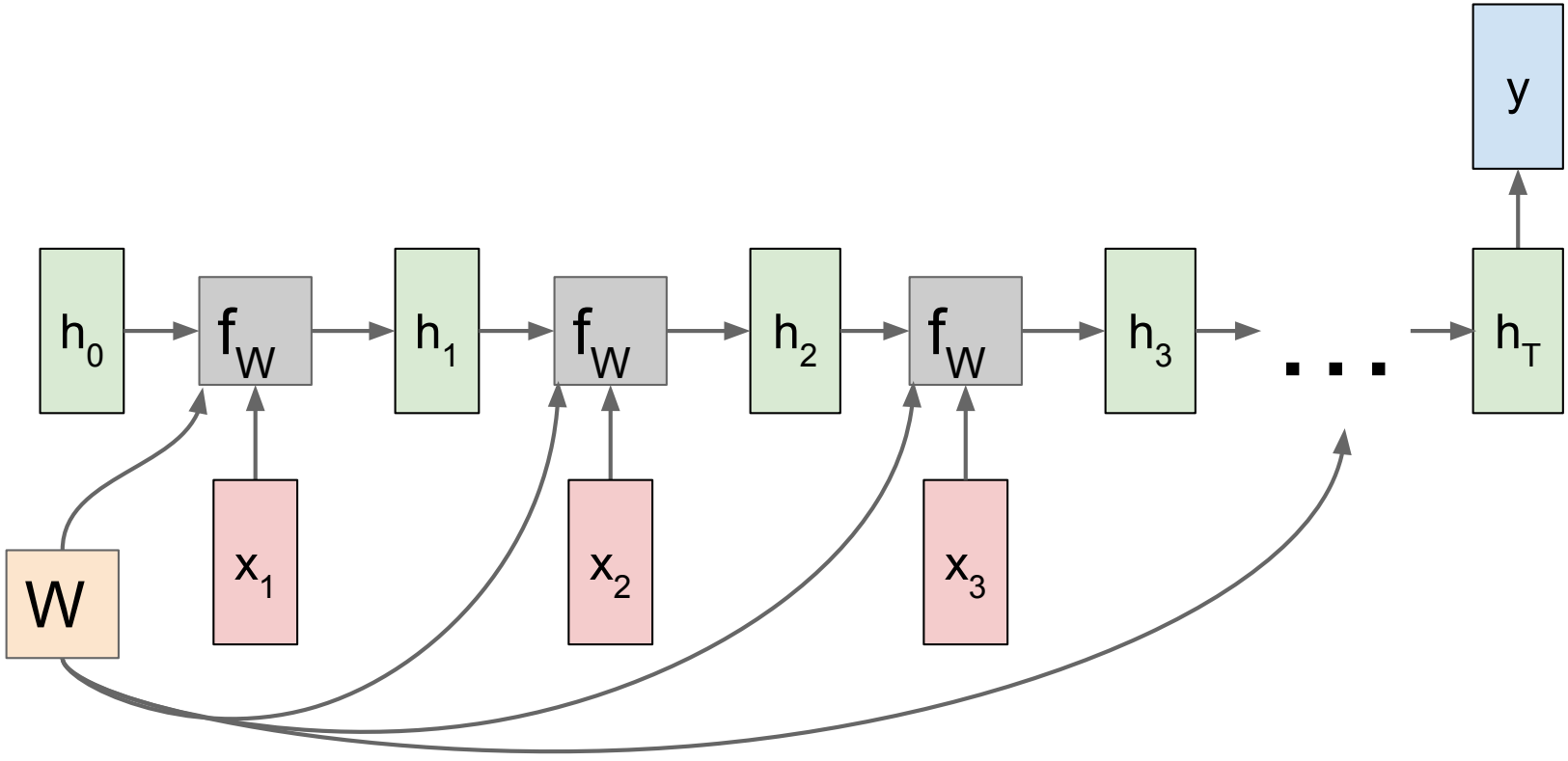
RNN: Computational Graph: Many to Many



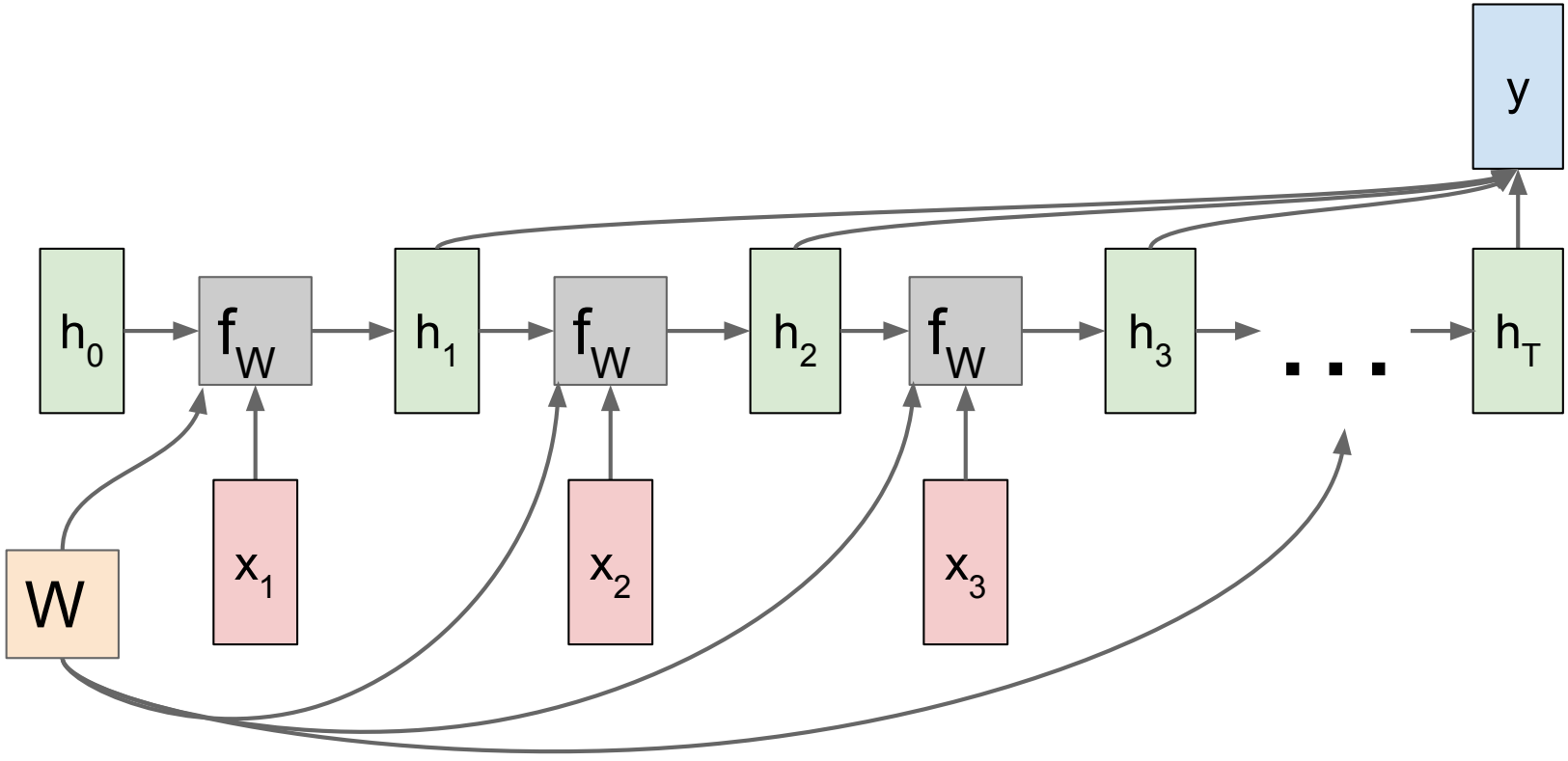
RNN: Computational Graph: Many to Many



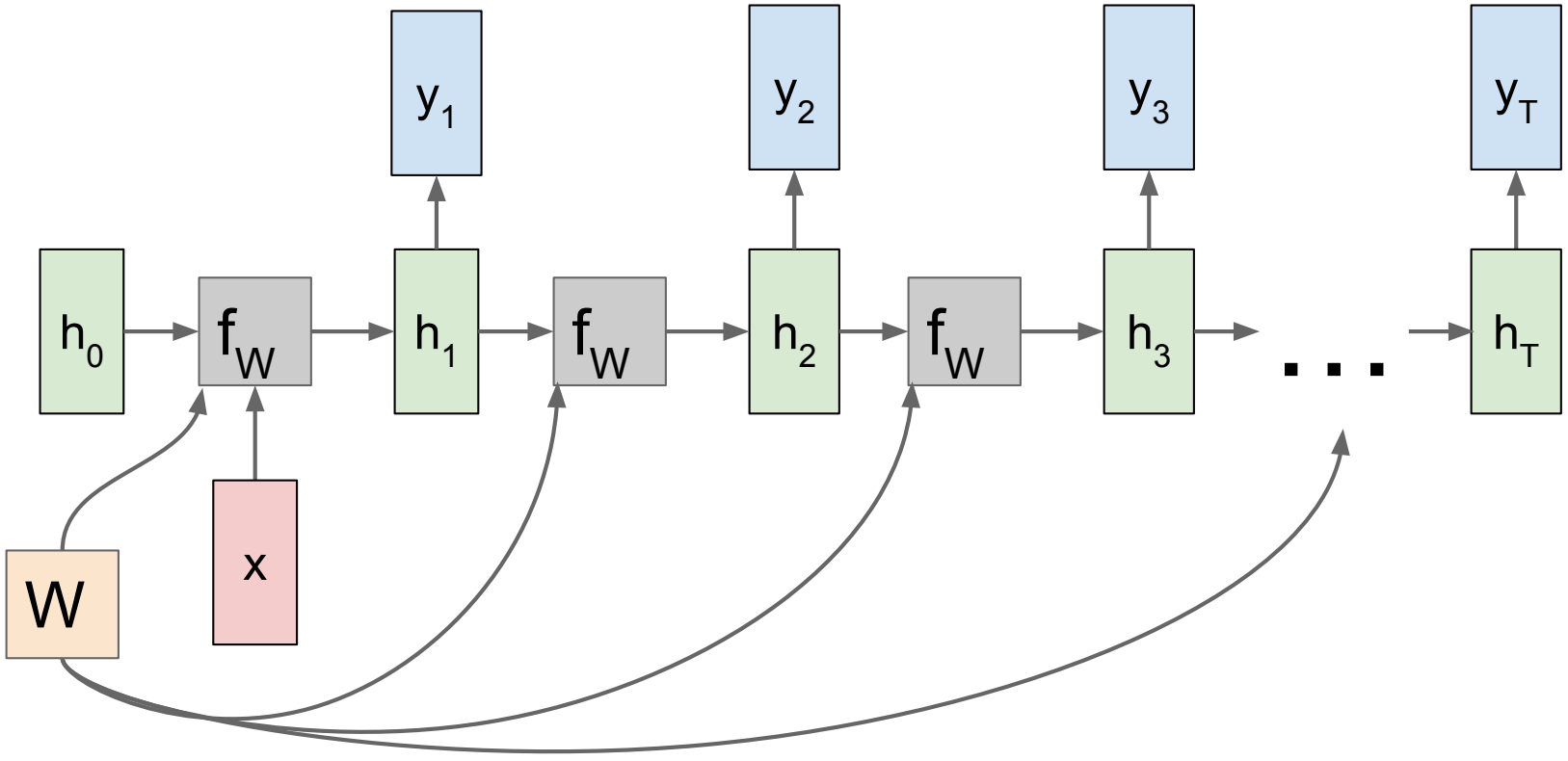
RNN: Computational Graph: Many to One



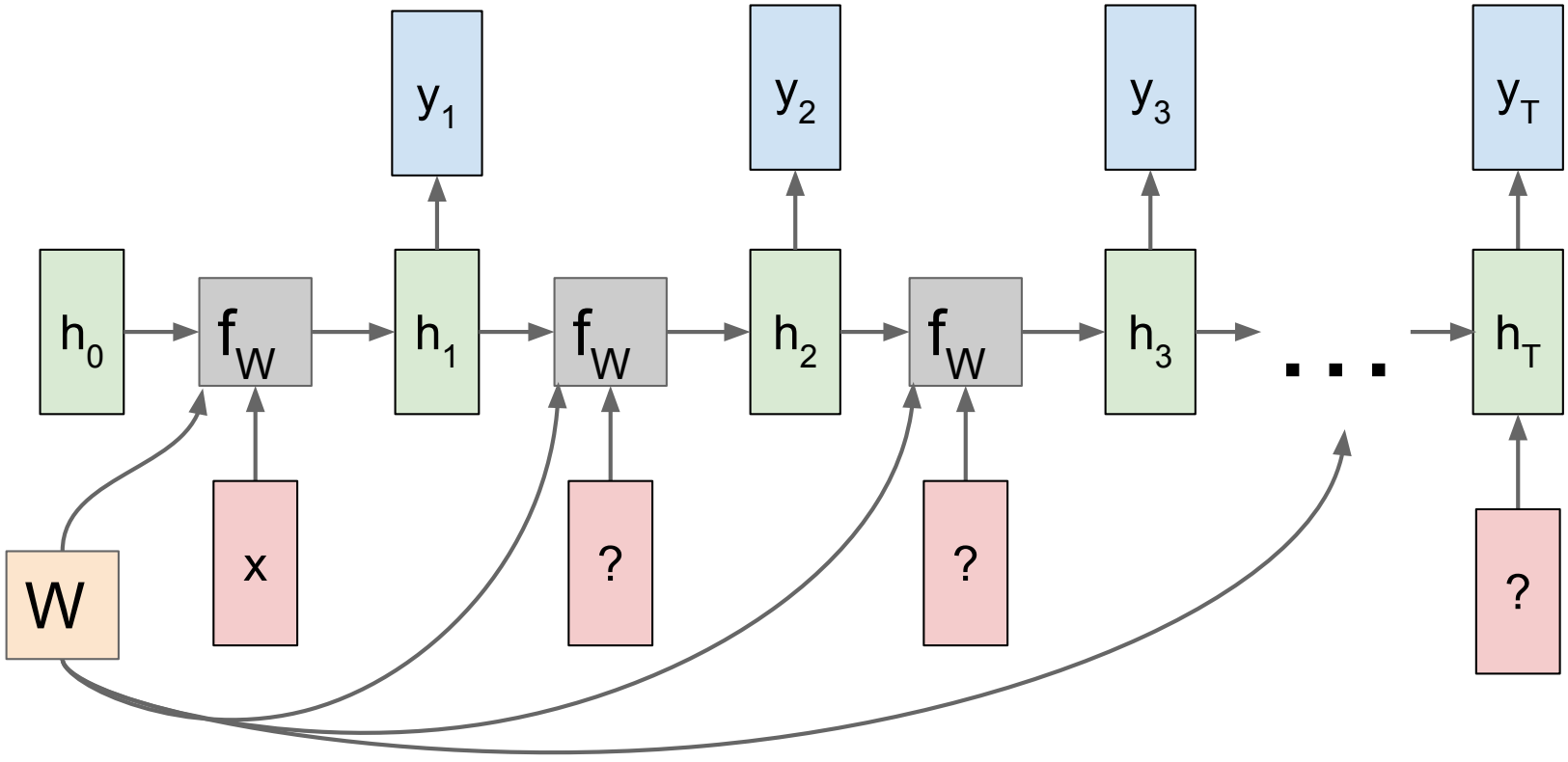
RNN: Computational Graph: Many to One



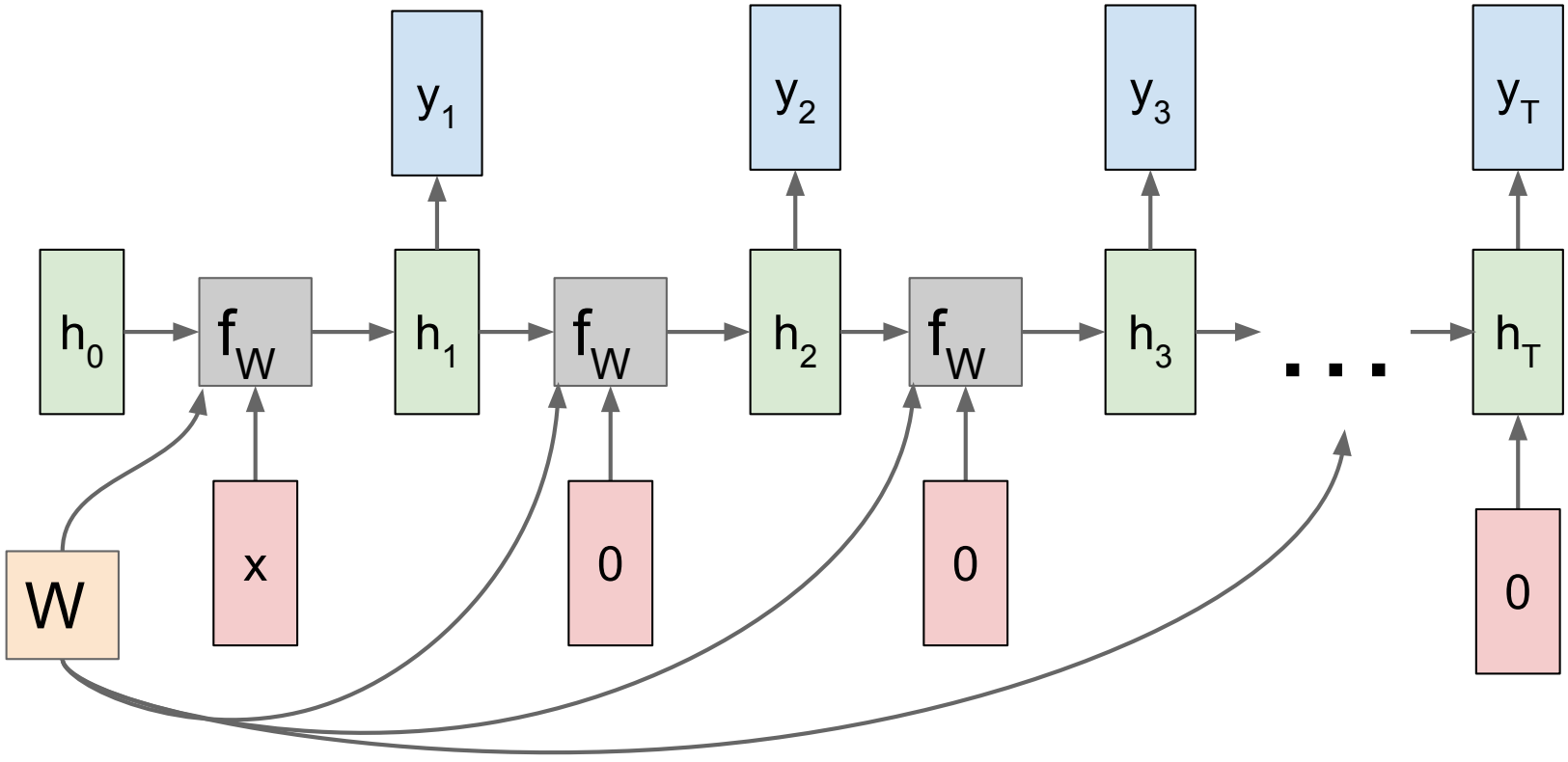
RNN: Computational Graph: One to Many



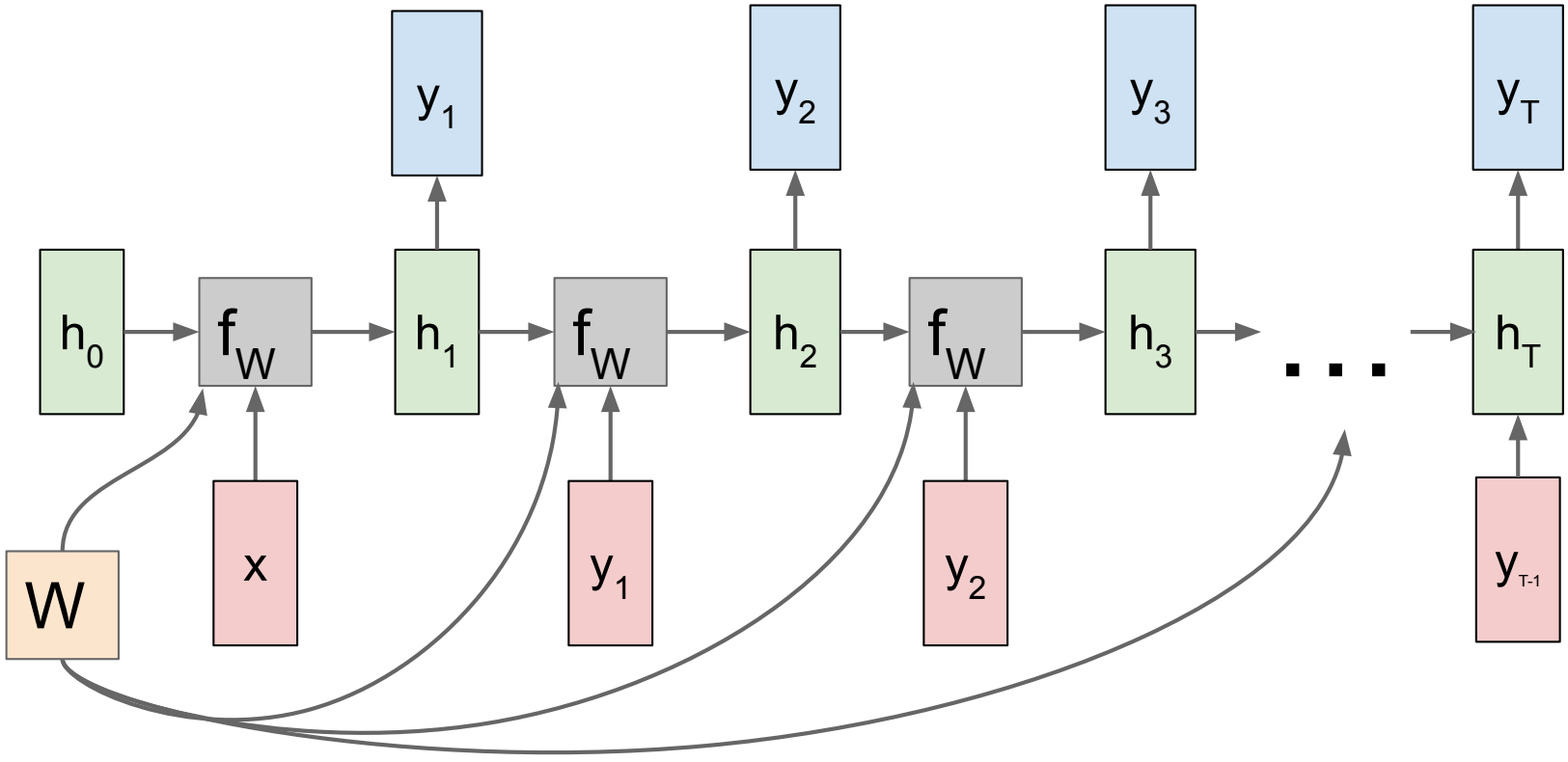
RNN: Computational Graph: One to Many



RNN: Computational Graph: One to Many

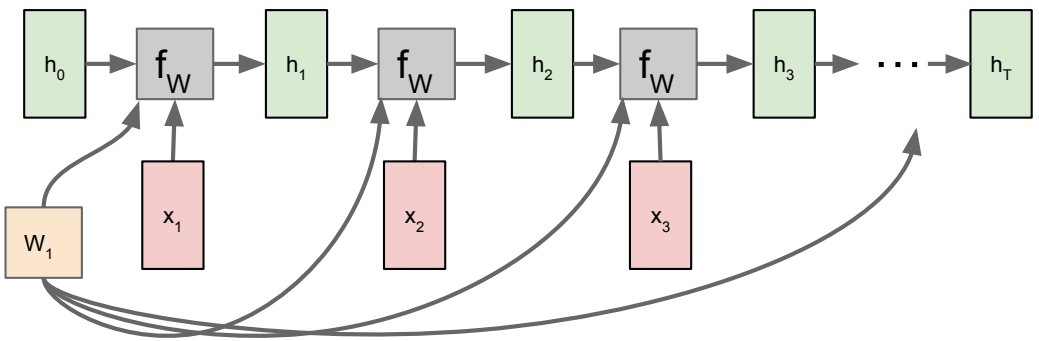


RNN: Computational Graph: One to Many



Sequence to Sequence: Many-to-one + one-to-many

Many to one: Encode input sequence in a single vector

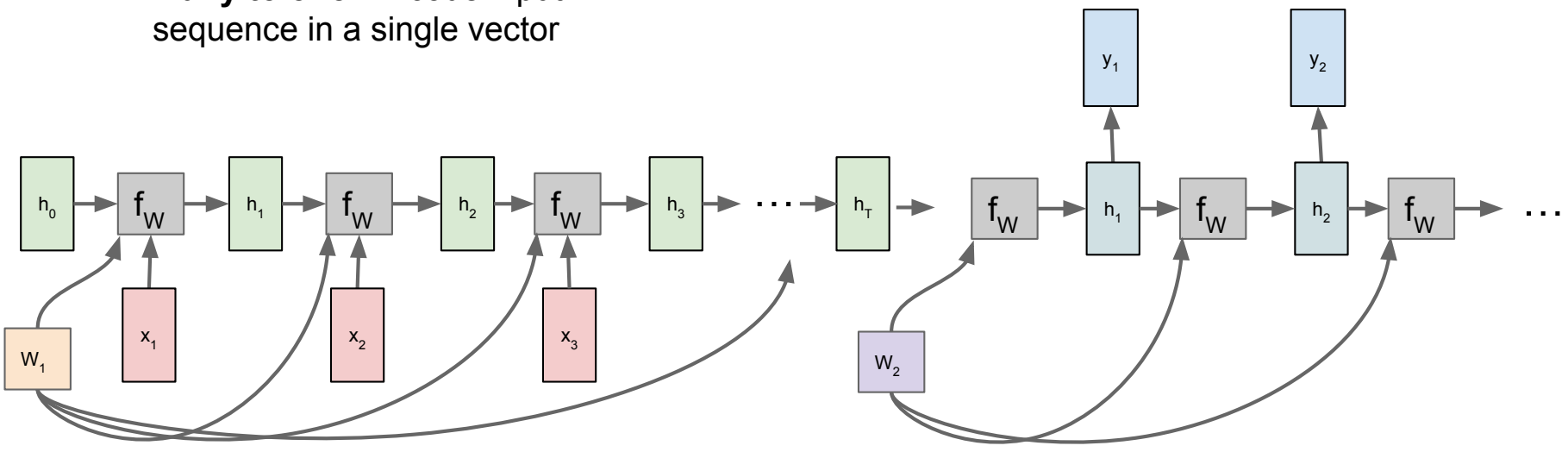


Sutskever et al, “Sequence to Sequence Learning with Neural Networks”, NIPS 2014

Sequence to Sequence: Many-to-one + one-to-many

Many to one: Encode input sequence in a single vector

One to many: Produce output sequence from single input vector



Sutskever et al, “Sequence to Sequence Learning with Neural Networks”, NIPS 2014

Recurrent neural network

Problems

- During training of long sequences the weights and gradients could increase and be unstable.
- With long sequences sequence, it will gradually forget the first inputs in the sequence.
- Due to the transformations that the data goes through when traversing an RNN, some information is lost at each time step. After a while, the RNN's state contains virtually no trace of the first inputs.

Solution - exploding gradients

- Reduce this risk by using a smaller learning rate
- Use a saturating activation function like the hyperbolic tangent (this explains why it is the default)
- Use Layer Normalization; implemented across features; it learns a scale and offset parameter for each input

Recurrent neural network

Solution - forgetfulness

- The Long Short-Term Memory (LSTM) cell was proposed in 1997 Hochreiter & Schmidhuber (1997) and gradually improved over the years by several researchers, such as Sak et al. (2014) and Zaremba et al. (2015).
- It can be used very much like a basic cell
- Performs much better and training converges faster
- It can detect long-term dependencies in the data

LSTM in keras/tensorflow

```
model = keras.models.Sequential([
keras.layers.LSTM(20, return_sequences=True, input_shape=[None, 1]),
keras.layers.LSTM(20, return_sequences=True),
keras.layers.TimeDistributed(keras.layers.Dense(10))
])
```

Recurrent neural network - LSTM

Key idea

The key idea is that the network can learn what to store in the long-term state, what to throw away, and what to read from it.

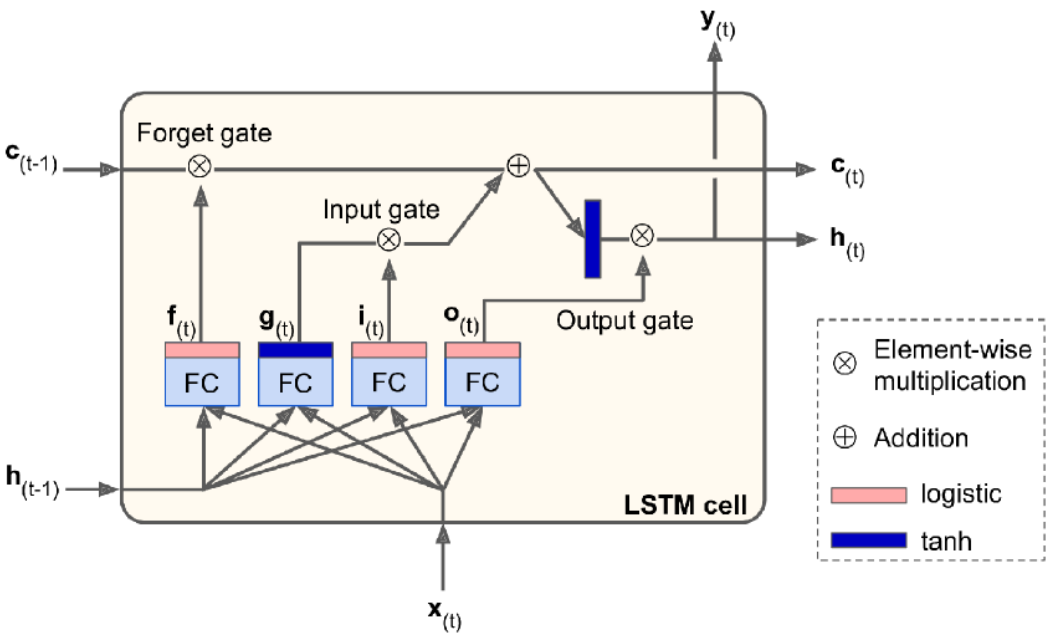


Figure 7: LSTM cell (Géron 2017)

Recurrent neural network - LSTM

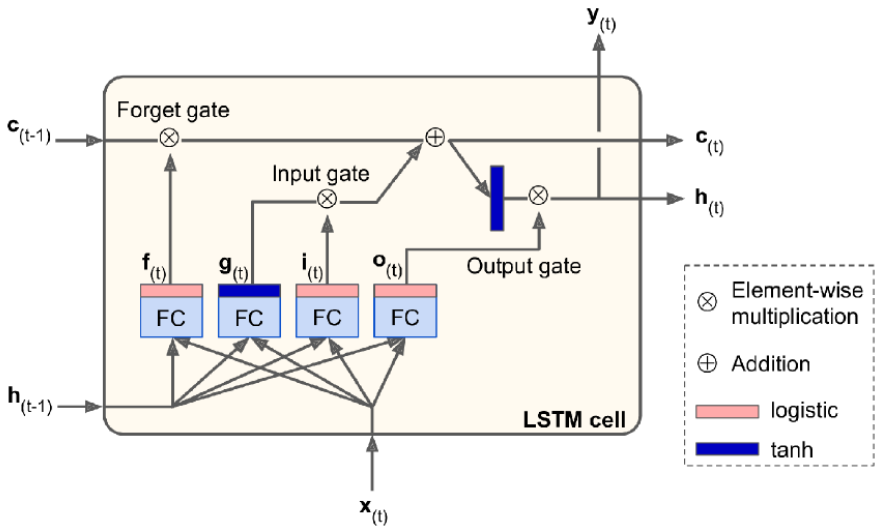


Figure 8: LSTM cell (Géron 2017)

General operation (see Figure 8)

- Long term state $c_{(t-t)}$ traverses through network (left to right)
- Passes through forget gate; drops some memory & adds new memory selected by input gate
- Long-term state is copied and passed through the $\tanh$ function, and then the result is filtered by the output gate.
- This produces the short-term state $h_{(t)}$; i.e. the cell's output for this time step,$y_{(t)}$

Recurrent neural network - LSTM

Overview

Current input vector $x_{(t)}$ and the previous short-term state $h_{(t-1)}$ are fed to four different fully connected layers.

1. Main layer is the one that outputs $g_{(t)}$. It has the usual role of analyzing the current inputs $x_{(t)}$ and the previous (short-term) state $h_{(t-1)}$.
2. The three other layers are gate controllers; they use the logistic activation function, their outputs range from 0 to 1.
3. Respective outputs are fed to element-wise multiplication operations; a zero closes the gate & a "one" opens it.
   - The forget gate (controlled by $f_{(t)}$) controls which parts of the long-term state should be erased.
   - The input gate (controlled by $i_{(t)}$) controls which parts of $g_{(t)}$ should be added to the long-term state.
   - Finally, the output gate (controlled by $o_{(t)}$) controls which parts of the long-term state should be read and output at this time step, both to $h_{(t)}$ and to $y_{(t)}$.

Recurrent neural network - LSTM

Computations of LSTM

$$
\begin{aligned}
i_{(t)} &= \sigma(W_{xi}^T x_{(t)} + W_{hi}^T h_{(t-1)} + b_i) \\
f_{(t)} &= \sigma(W_{xf}^T x_{(t)} + W_{hf}^T h_{(t-1)} + b_f) \\
o_{(t)} &= \sigma(W_{xo}^T x_{(t)} + W_{ho}^T h_{(t-1)} + b_o) \\
g_{(t)} &= \tanh(W_{xg}^T x_{(t)} + W_{hg}^T h_{(t-1)} + b_g) \\
c_{(t)} &= f_{(t)} \otimes c_{(t-1)} + i_{(t)} \otimes g_{(t)} \\
y_{(t)} &= h_{(t)} = 0_{(t)} \otimes \tanh(c_{(t)})
\end{aligned}
$$

In these equations:

- $W_{xi}, W_{xf}, W_{xo}, W_{xg}$ are weight matrices of each of the 4 FC layers for their connection to input vector $x_{(t)}$
- $W_{hi}, W_{hf}, W_{ho}, W_{hg}$ are weight matrices of the 4 FC layers for their connection to previous short-term state $h_{(t-1)}$
- b_i, b_f, b_o, b_g are the bias term for each of the 4 FC layers.

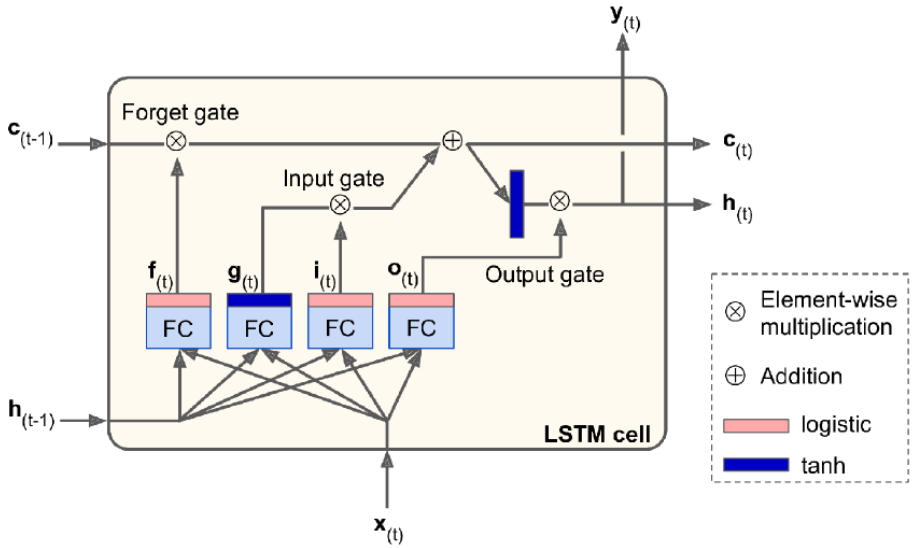


Figure 9: LSTM cell (Géron 2017)

Recurrent neural network - GRU

Gated Recurrent Neural Network

1. Another recurrent unit similar to the LSTM is the Gated Recurrent Unit (GRU)
2. GRU has a reset gate and an update gate, similar to the forget/input gates in the LSTM unit.
3. The major difference is that the GRU fully exposes its memory content using only leaky integration (but with an adaptive time constant controlled by the update gate).
4. The GRU was inspired by the LSTM unit but is considered simpler to compute and implement.

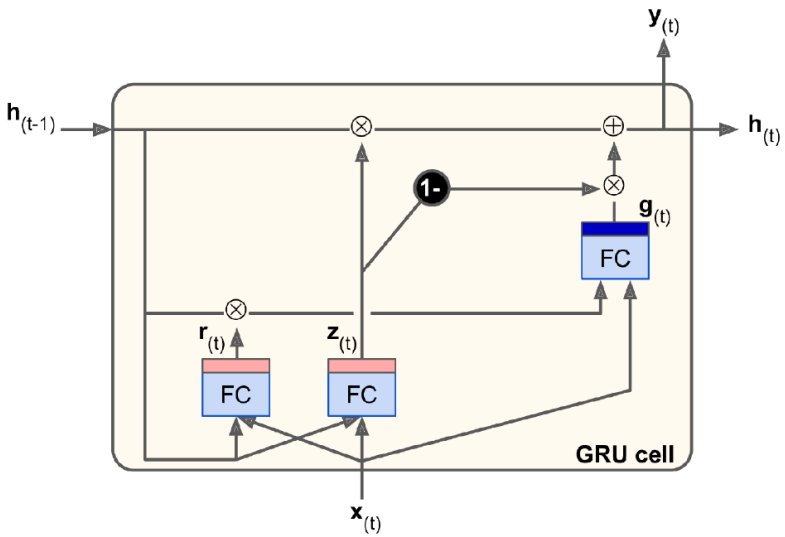


Figure 10: GRU cell (Géron 2017)

Recurrent neural network - GRU

Computations of GRU

$$z_{(t)} = \sigma(W_{xz}^T x_{(t)} + W_{hz}^T h_{(t-1)} + b_z)$$
$$r_{(t)} = \sigma(W_{xr}^T x_{(t)} + W_{hr}^T h_{(t-1)} + b_r)$$
$$g_{(t)} = \tanh(W_{xg}^T x_{(t)} + W_{hg}^T (r_{(t)} \otimes h_{(t-1)}) + b_g)$$
$$h_{(t)} = z_{(t)} \otimes h_{(t-1)} + (1 - z_{(t)}) \otimes gc_{(t)}$$

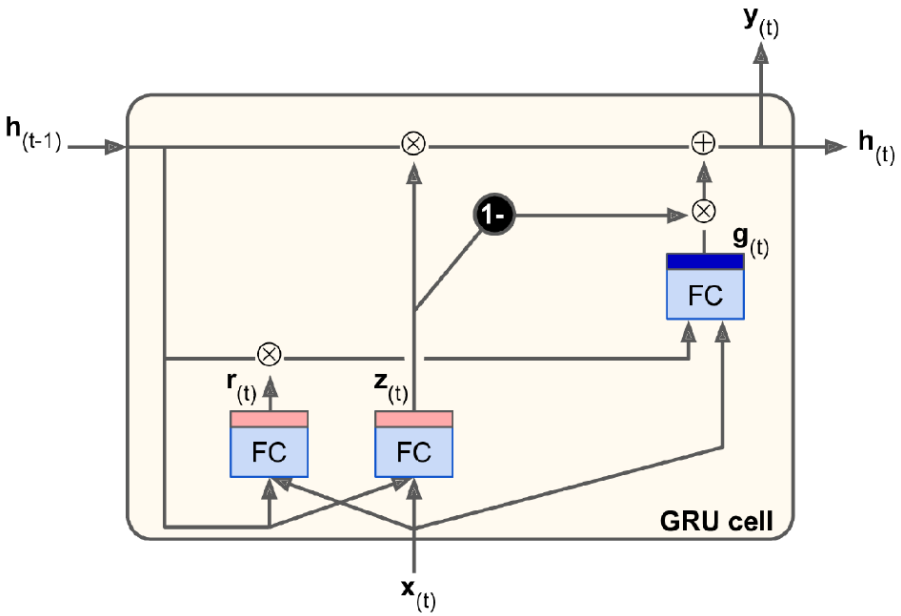


Figure 11: GRU cell (Géron 2017)

Recurrent neural network - LSTM & GRU

End note

1. Despite advantage of LSTM and GRU over RNN; i.e. dealing with much longer sequences; they have fairly short-term memory and difficulty learning long-term patterns in sequences of 100 time steps or more.
2. One solution is to shorten the input sequence; use a 1-D CNN as pre-processor

Keras implementation of 1-D CNN solution

```
model = keras.models.Sequential([
keras.layers.Conv1D(filters=20, kernel_size=4, strides=2,
padding="valid",
input_shape=[None, 1]),
keras.layers.GRU(20, return_sequences=True),
keras.layers.GRU(20, return_sequences=True),
keras.layers.TimeDistributed(keras.layers.Dense(10))
])
```

Bibliography

Chollet, F. (2021), *Deep Learning with Python*, 2nd edn, Manning Publications Co., Shelter Island, NY, USA.

Géron, A. (2017), *Hands-on Machine Learning with Scikit-Learn and TensorFlow*, O'Reilly Media, Inc., CA, USA.

Hochreiter, S. & Schmidhuber, J. (1997), 'Long short-term memory', *Neural Computation* **9**(8), 1735–1780.

Sak, H., Senior, A. & Beaufays, F. (2014), Long short-term memory based recurrent neural network architectures for large vocabulary speech recognition, Technical Report arXiv:1512.03385 [cs.NE], ArXiV.
URL: *https://arxiv.org/pdf/1402.1128.pdf*

Zaremba, W., Sutskever, I. & Vinyals, O. (2015), Recurrent neural network regularization, Technical Report arXiv:1409.2329 [cs.NE], ArXiV.
URL: *https://arxiv.org/pdf/1409.2329.pdf*