References for the next pages

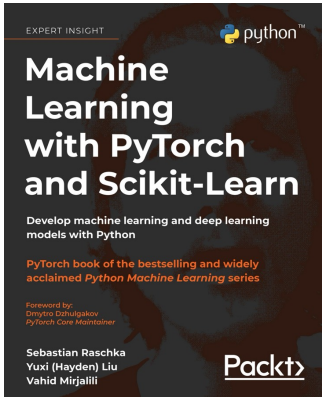
Machine Learning with PyTorch and Scikit-Learn,

S.Raschka, Y.Liu & V.Mirjalili

Packt Publishing (2022)

ISBN 978-1-80181-931-2

Highly recommended, but uses Python / Pytorch



Deep Learning: A Visual Approach

Andrew Glassner

No Starch Press (2021)

ISBN 978-1-71850-072-3

Extremely recommended, conceptual (no code)

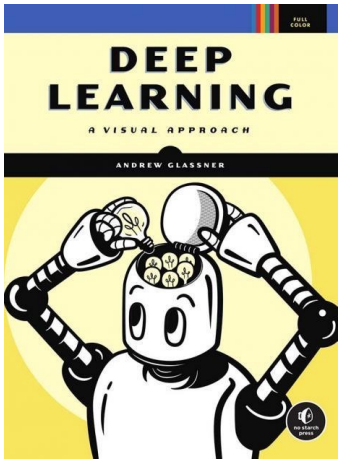


Fig. 13-5 in Glassner, an artificial neuron

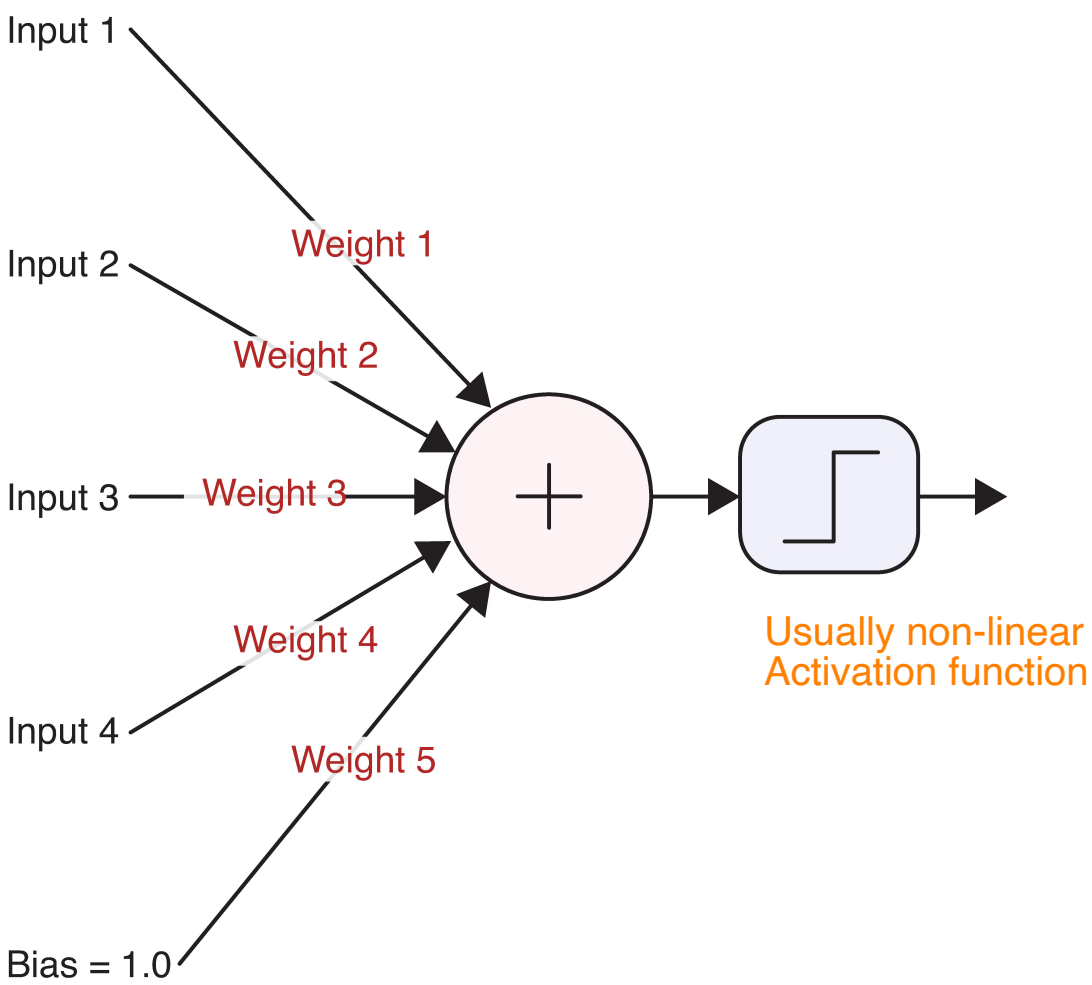
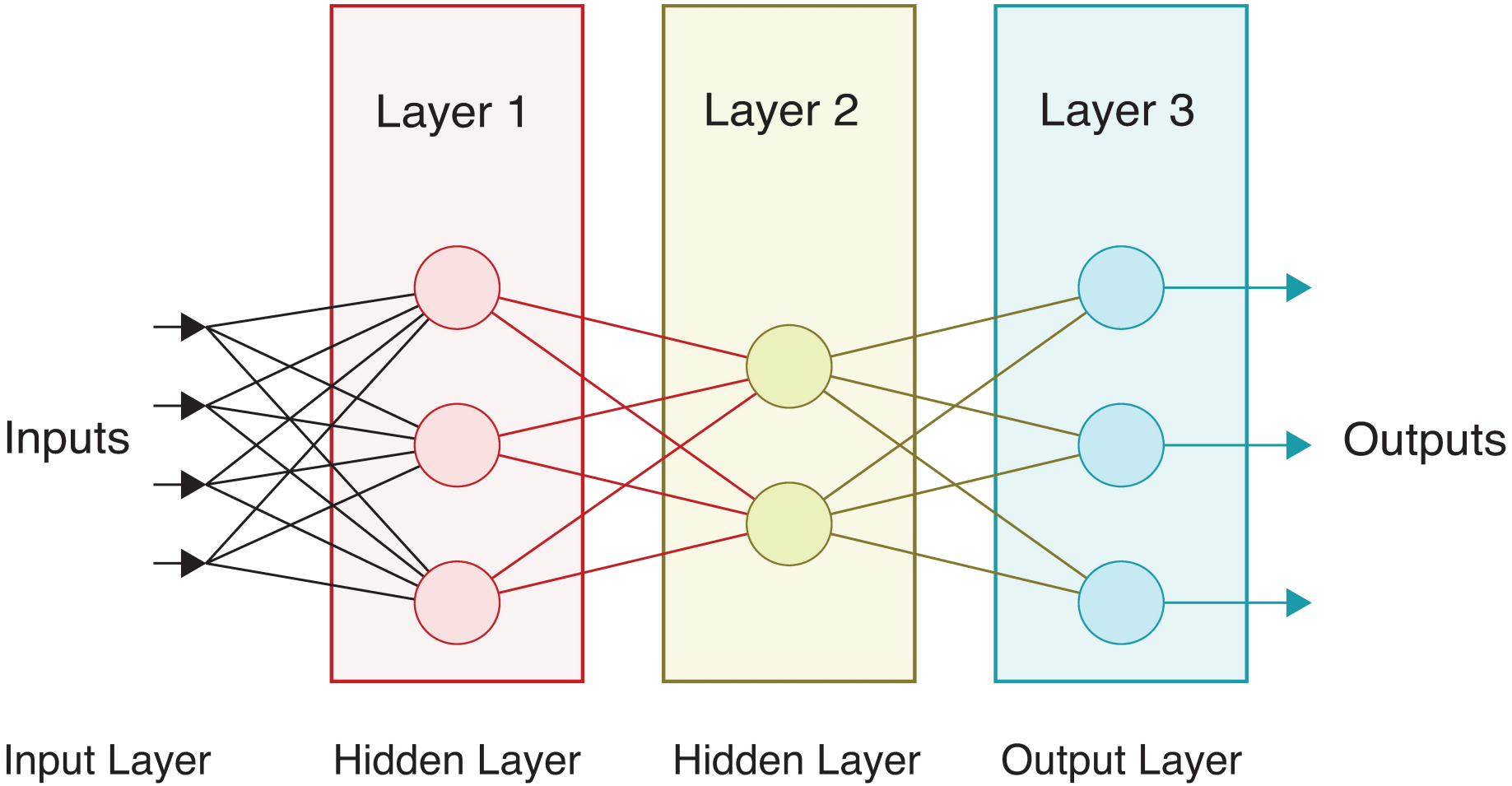
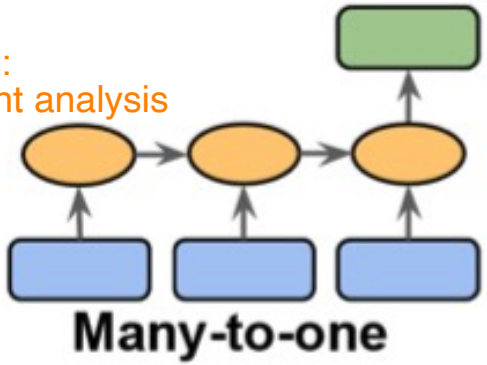
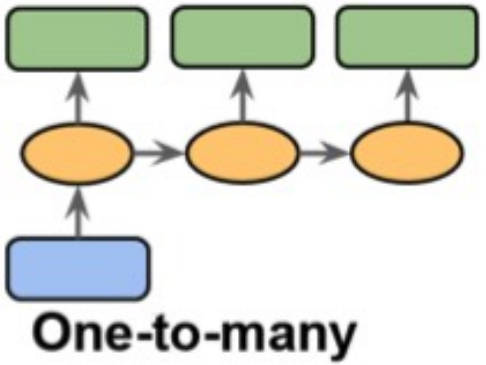


Fig. 13-11 in Glassner, a deep learning neural network (with 3 layers).
This is an example of a dense feedforward neural network, also called a multilayer perceptron (MLP)



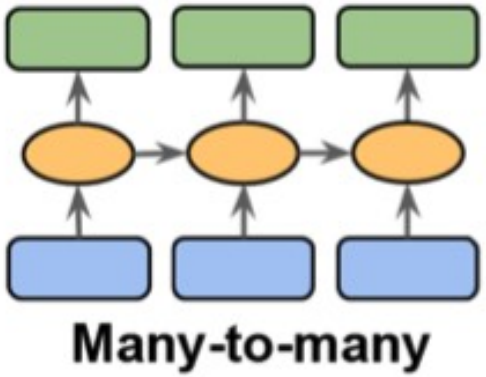
Example:
Sentiment analysis





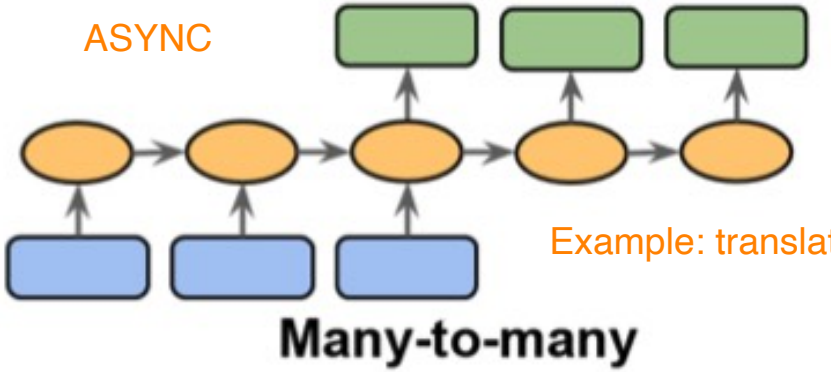
Example:
Image Labeling

SYNC



Example:
video frame labeling,
DNA

ASYNC



Example: translation, generative text

Figure 15.2: The most common sequencing tasks

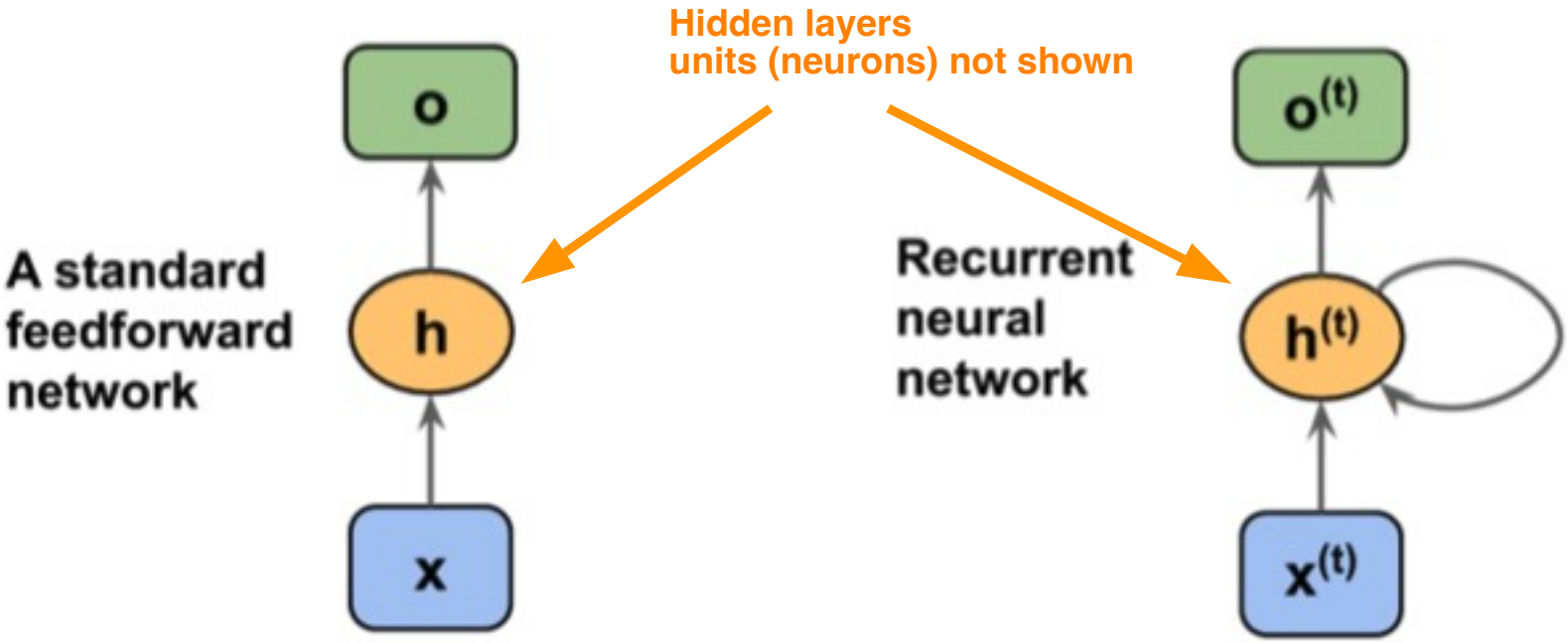


Figure 15.3: The dataflow of a standard feedforward NN and an RNN

Fig. 19-11 in Glassner, a recurrent neural cell

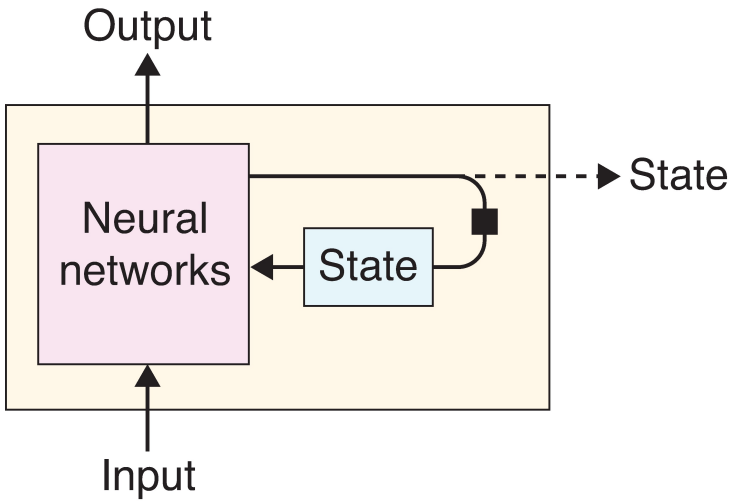
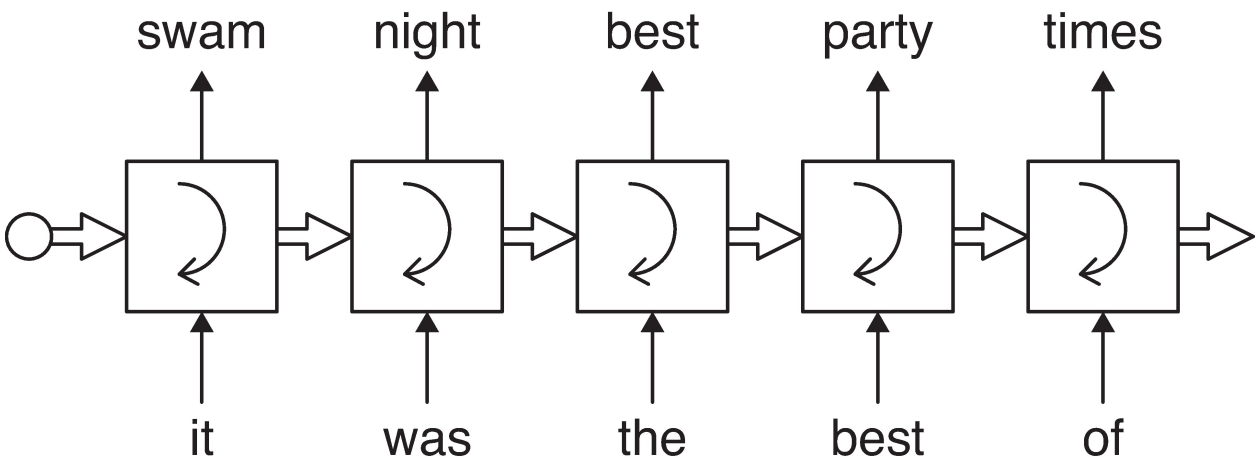
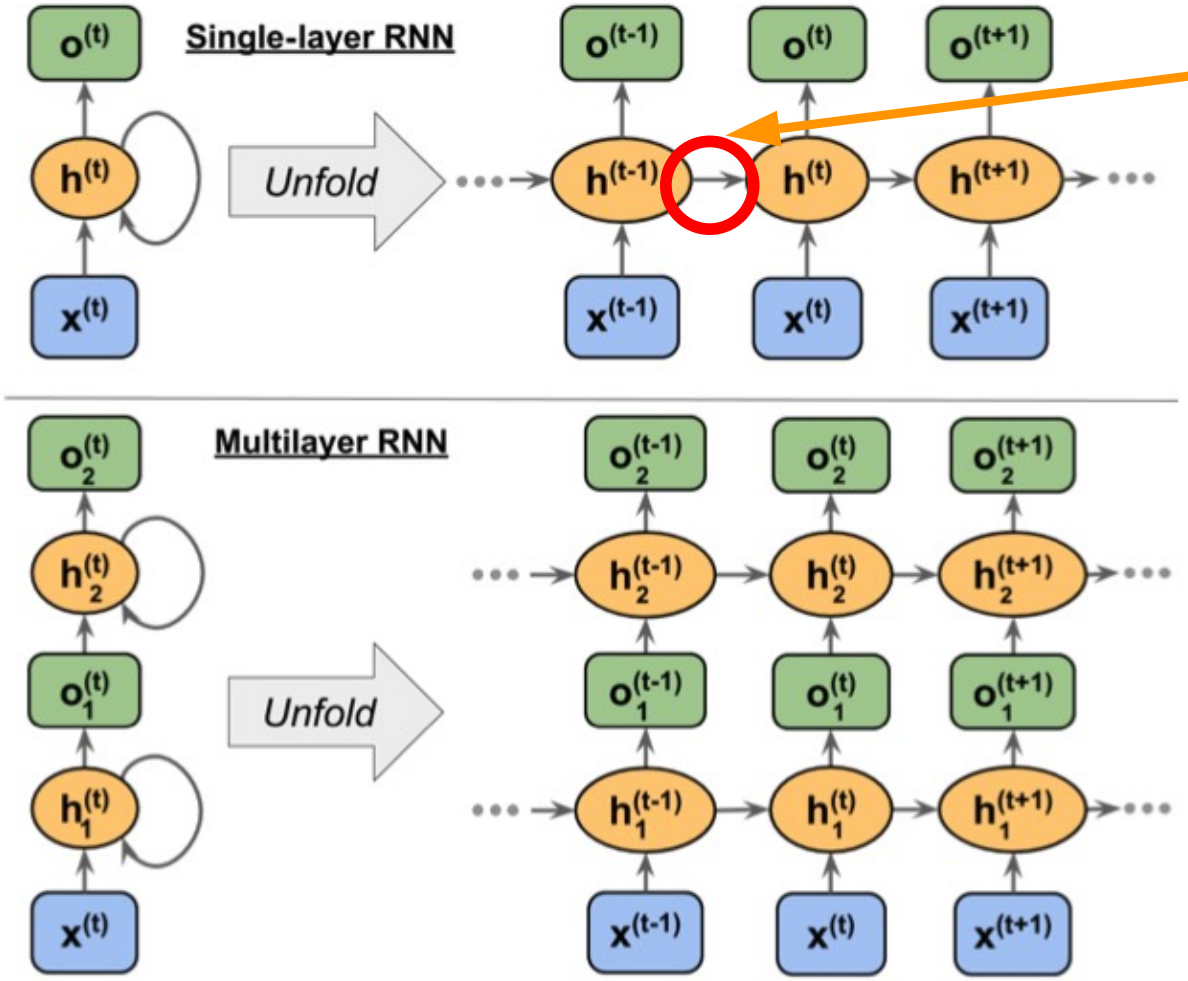


Fig. 19-13 in Glassner, an unrolled recurrent neural network, used for NLP tasks (next word prediction)



in an RNN, the hidden layer receives its input from both the input layer of the current time step and the hidden layer from the previous time step.



A new kind of input:

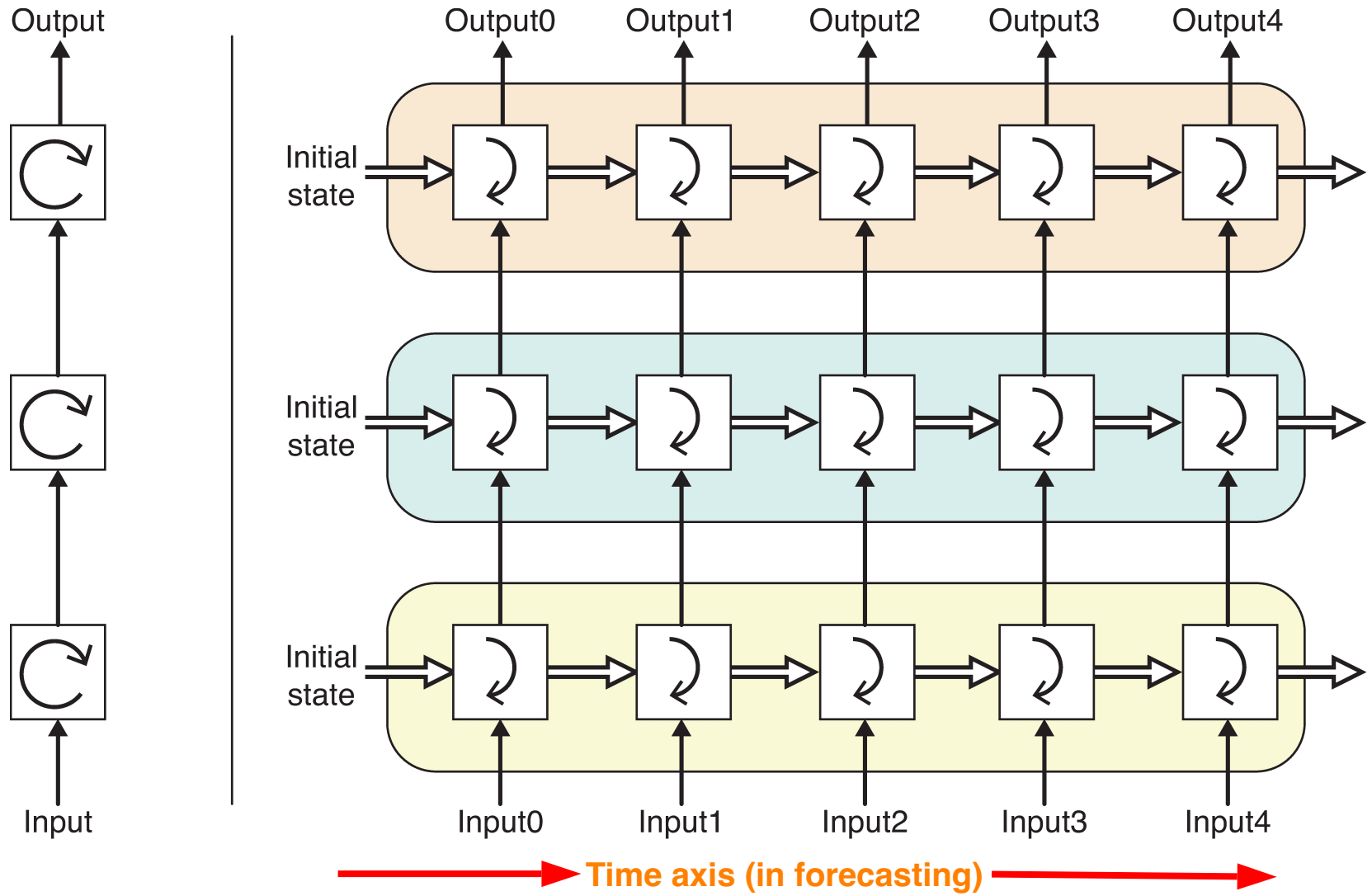
Allows the RNN to have a memory!

Multilayer network:

return sequences is True for "deep" layers

Figure 15.4: Examples of an RNN with one and two hidden layers

Fig. 19-19 in Glassner, a deep RNN (3 layers) in rolled(left) and unrolled form.



Different types of weight matrices in RNNs

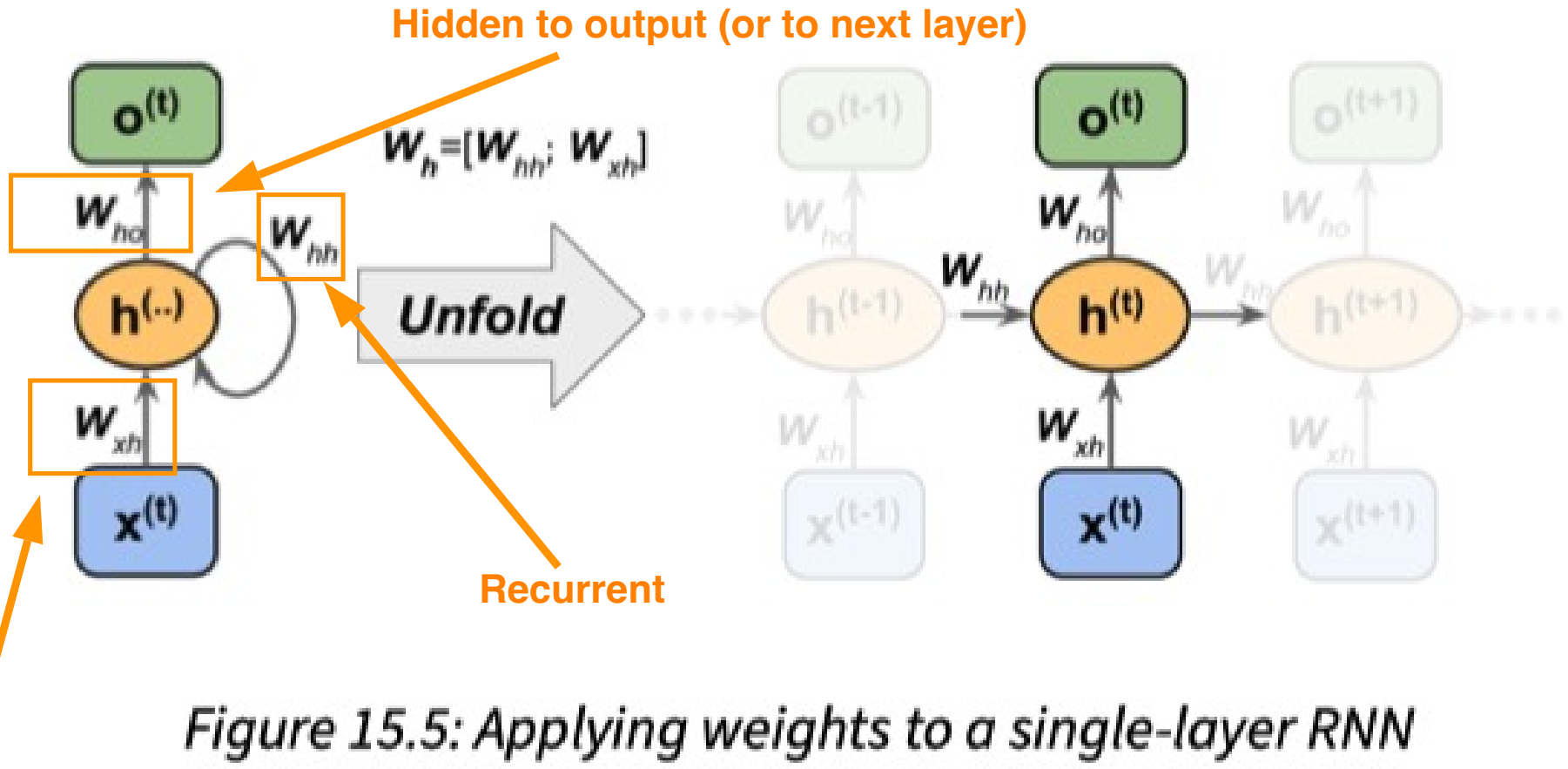


Figure 15.5: Applying weights to a single-layer RNN

https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=58337

Backpropagation Through Time: What It Does and How to Do It

PAUL J. WERBOS

1990 PROCEEDINGS OF THE IEEE

Backpropagation is now the most widely used tool in the field of artificial neural networks. At the core of backpropagation is a method for calculating derivatives exactly and efficiently in any large system made up of elementary subsystems or calculations which are represented by known, differentiable functions; thus, backpropagation has many applications which do not involve neural networks as such.

This paper first reviews basic backpropagation, a simple method which is now being widely used in areas like pattern recognition and fault diagnosis. Next, it presents the basic equations for backpropagation through time, and discusses applications to areas like pattern recognition involving dynamic systems, systems identification, and control. Finally, it describes further extensions of this method, to deal with systems other than neural networks, systems involving simultaneous equations or true recurrent networks, and other practical issues which arise with this method. Pseudocode is provided to clarify the algorithms. The chain rule for ordered derivatives—the theorem which underlies backpropagation—is briefly discussed.

I. Introduction

Backpropagation through time is a very powerful tool, with applications to pattern recognition, dynamic modeling, sensitivity analysis, and the control of systems over time, among others. It can be applied to neural networks, to econometric models, to fuzzy logic structures, to fluid dynamics models, and to almost any system built up from elementary subsystems or calculations. The one serious constraint is that the elementary subsystems must be represented by functions known to the user, functions which are both continuous and differentiable (i.e., possess derivatives). For example, the first practical application of backpropagation was for estimating a dynamic model to predict nationalism and social communications in 1974 [1].

Unfortunately, the most general formulation of backpropagation can only be used by those who are willing to work out the mathematics of their particular application. This paper will mainly describe a simpler version of backpropagation, which can be translated into computer code and applied directly by neural network users.

Section II will review the simplest and most widely used form of backpropagation, which may be called "basic backpropagation." The concepts here will already be familiar to those who have read the paper by Rumelhart, Hinton, and Williams [2] in the seminal book *Parallel Distributed Processing*, which played a pivotal role in the development of the field. (That book also acknowledged the prior work of Parker [3] and Le Cun [4], and the pivotal role of Charles Smith of the Systems Development Foundation.) This section will use new notation which adds a bit of generality and makes it easier to go on to complex applications in a rigorous manner. (The need for new notation may seem unnecessary to some, but for those who have to *apply* backpropagation to complex systems, it is essential.)

Section III will use the same notation to describe backpropagation through time. Backpropagation through time has been applied to concrete problems by a number of authors, including, at least, Watrous and Shastri [5], Sawai and Waibel *et al.* [6], Nguyen and Widrow [7], Jordan [8], Kawato [9], Elman and Zipser, Narendra [10], and myself [1], [11], [12], [15]. Section IV will discuss what is missing in this simplified discussion, and how to do better.

At its core, backpropagation is simply an efficient and exact method for calculating all the derivatives of a single target quantity (such as pattern classification error) with respect to a large set of input quantities (such as the parameters or weights in a classification rule). Backpropagation through time extends this method so that it applies to dynamic systems. This allows one to calculate the derivatives needed when optimizing an iterative analysis procedure, a neural network with memory, or a control system which maximizes performance over time.

II. Basic Backpropagation

A. The Supervised Learning Problem

Basic backpropagation is current the most popular method for performing the supervised learning task, which is symbolized in Fig. 1.

In supervised learning, we try to adapt an artificial neural network so that its actual outputs ($\hat{Y}$) come close to some target outputs (Y) for a training set which contains T patterns. The goal is to adapt the parameters of the network so that it performs well for patterns from outside the training set.

The main use of supervised learning today lies in pattern

Manuscript received September 12, 1989; revised March 15, 1990.
The author is with the National Science Foundation, 1800 G St. NW, Washington, DC 20550.
IEEE Log Number 9039172.

U.S. Government work not protected by U.S. copyright

The problem of gradient size in RNNs

On the difficulty of training recurrent neural networks by R. Pascanu, T. Mikolov, and Y. Bengio, 2012 (https://arxiv.org/pdf/1211.5063.pdf).

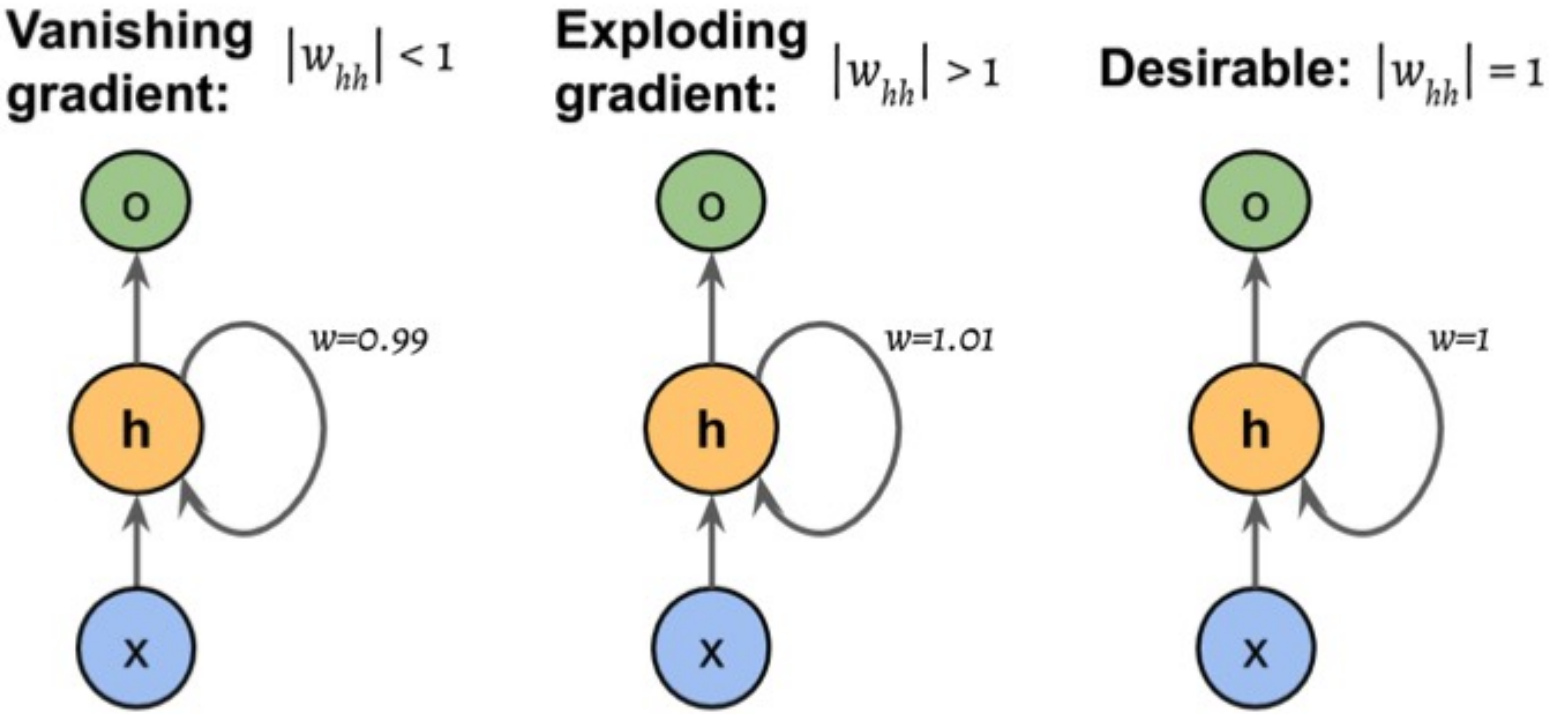


Figure 15.8: Problems in computing the gradients of the loss function

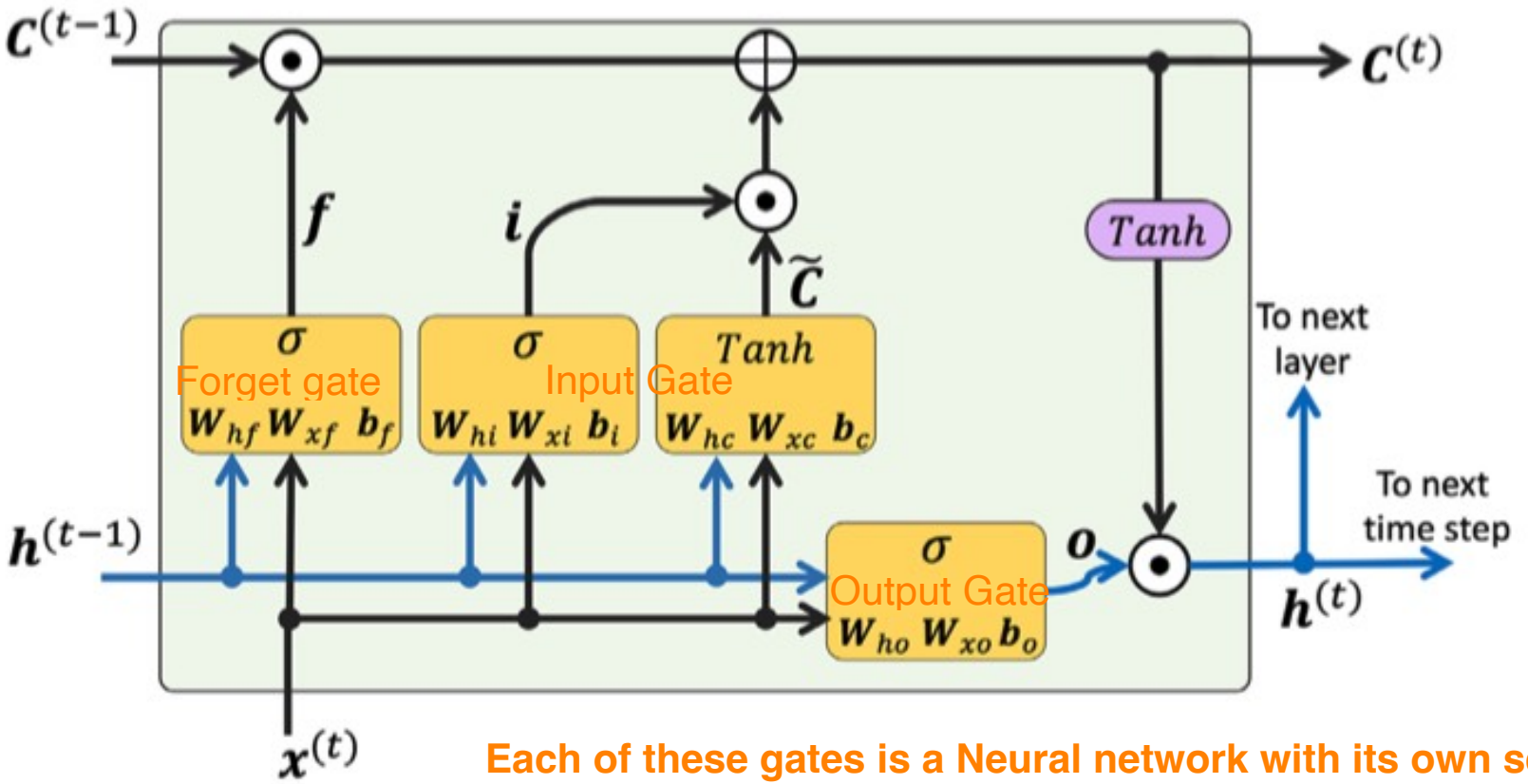
Alternatives:

- *Gradient clipping*
- *Truncated backpropagation*

- *LSTM* (Long Short-Term Memory by S. Hochreiter and J. Schmidhuber, Neural Computation, 9(8): 1735-1780, 1997

LSTM Cell (modern version)

Adds: gates + state



Each of these gates is a Neural network with its own set of weights and biases

Figure 15.9: The structure of an LSTM cell

Note:

· The LSTM cell essentially replaces the hidden cell of the RNN, but it adds a new recurrent edge, called the state C_t.

· Cell state from previous time step, C_(t–1), is used to get the current state C_t, without being multiplied directly by any weight factor (avoids vanishing or exploding gradient).

References for the next pages

Medium post: Animated RNN, LSTM and GRU by Raimi Karim

https://towardsdatascience.com/animated-rnn-lstm-and-gru-ef124d06cf45

Medium post: Illustrated Guide to LSTM's and GRU's: A step by step explanation by Michael Phi

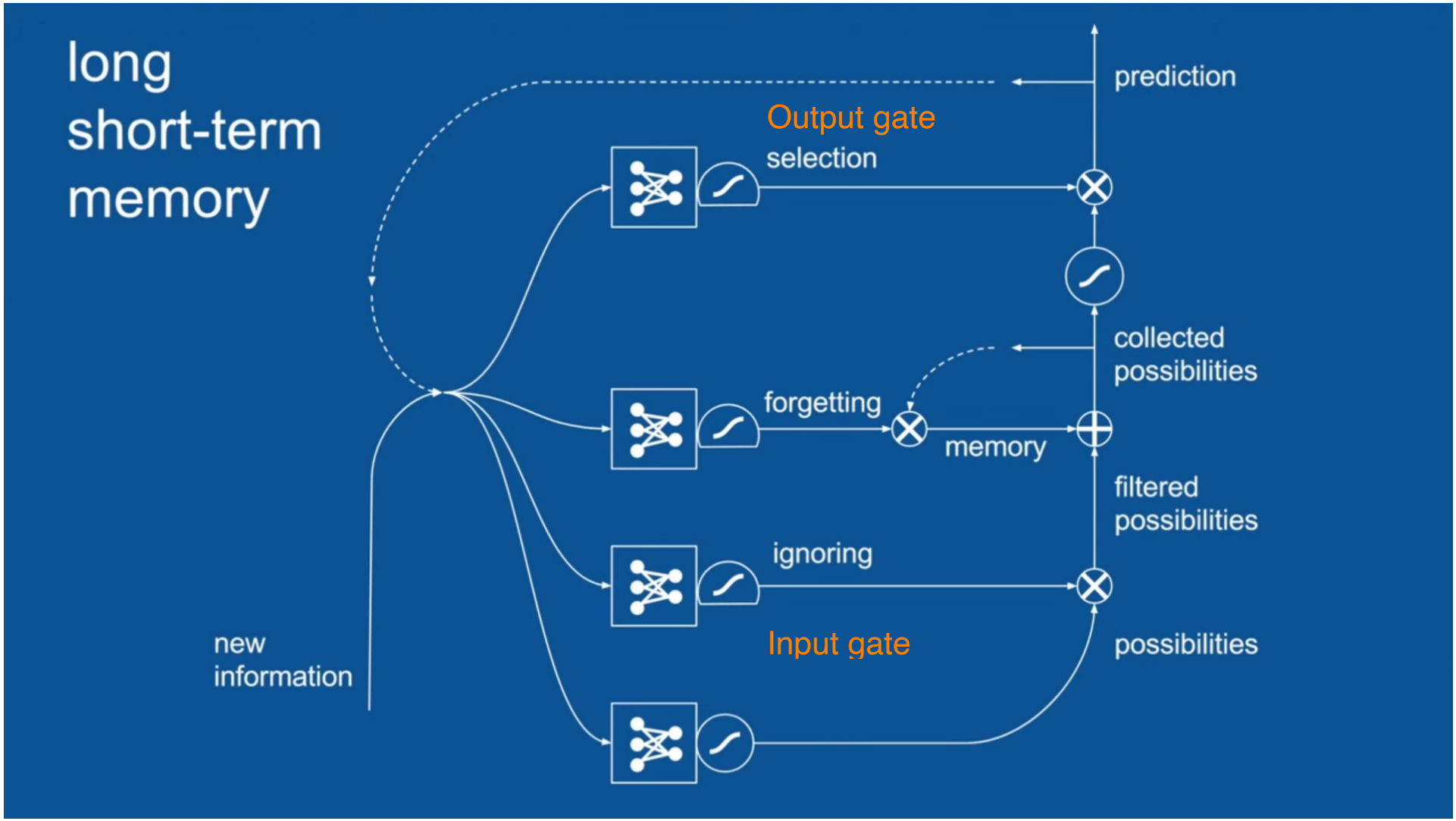
https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21

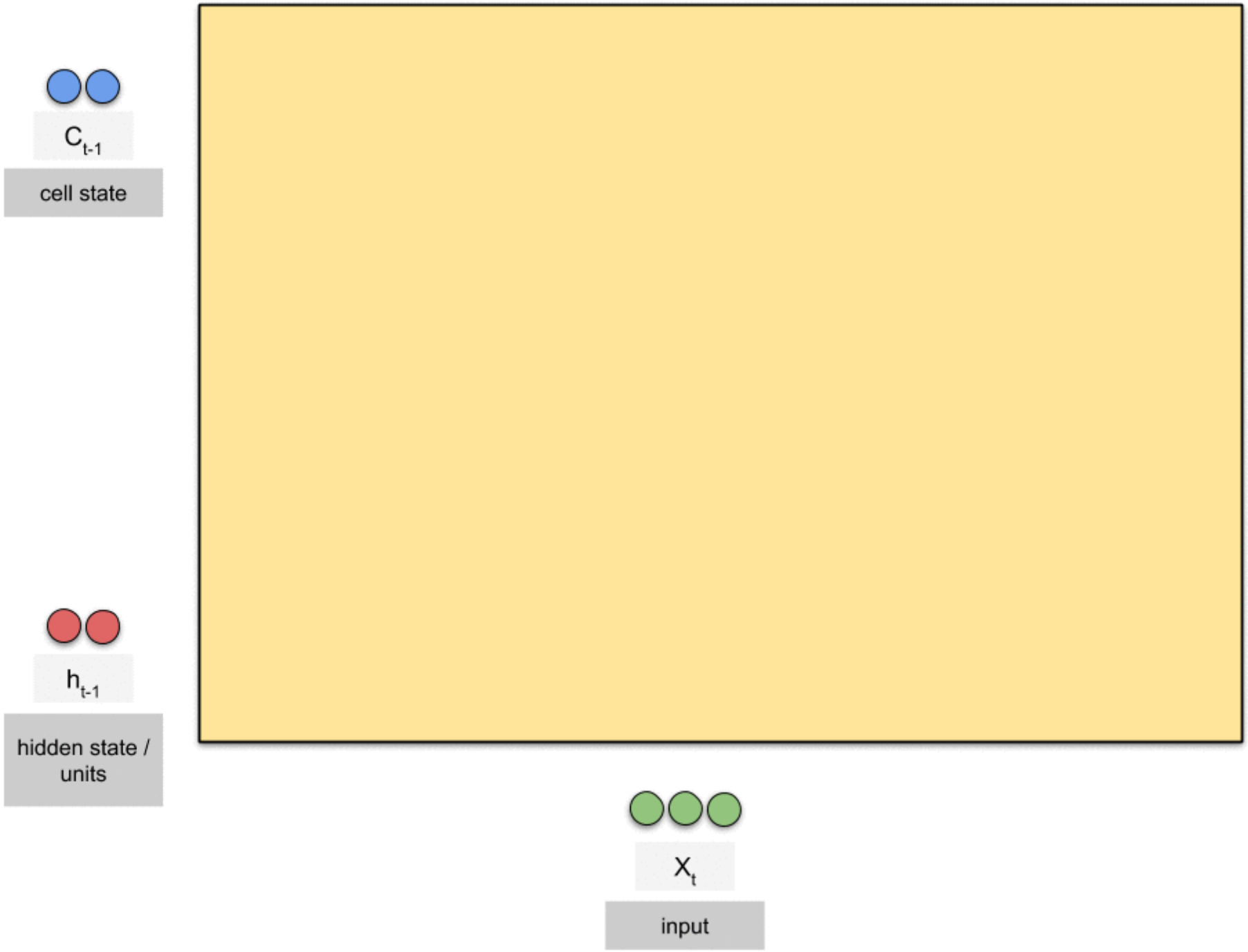
Youtube video: Recurrent Neural Networks (RNN) and Long Short-Term Memory (LSTM) by Brandon Rohrer

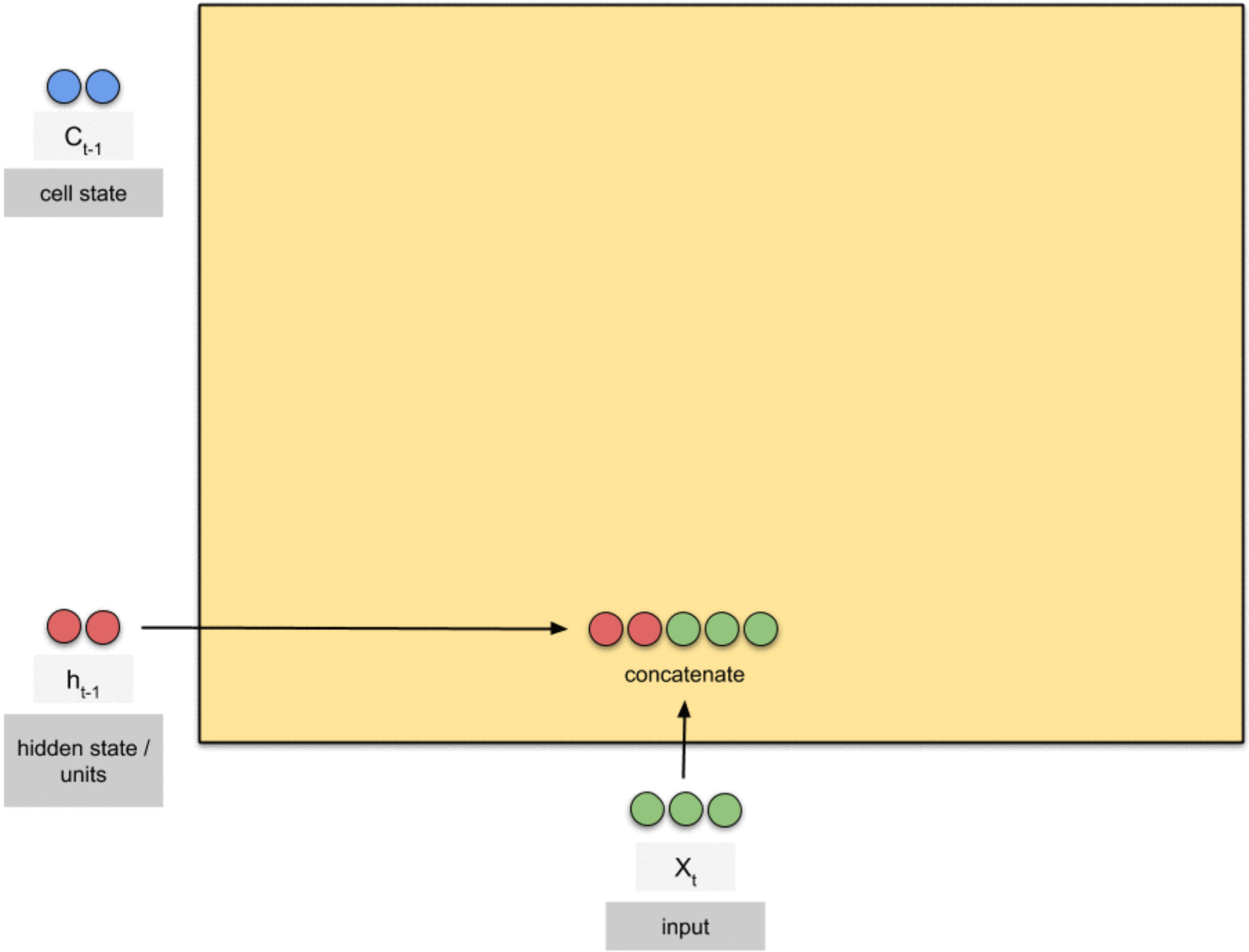
https://www.youtube.com/watch?v=WCUNPb-5EYI

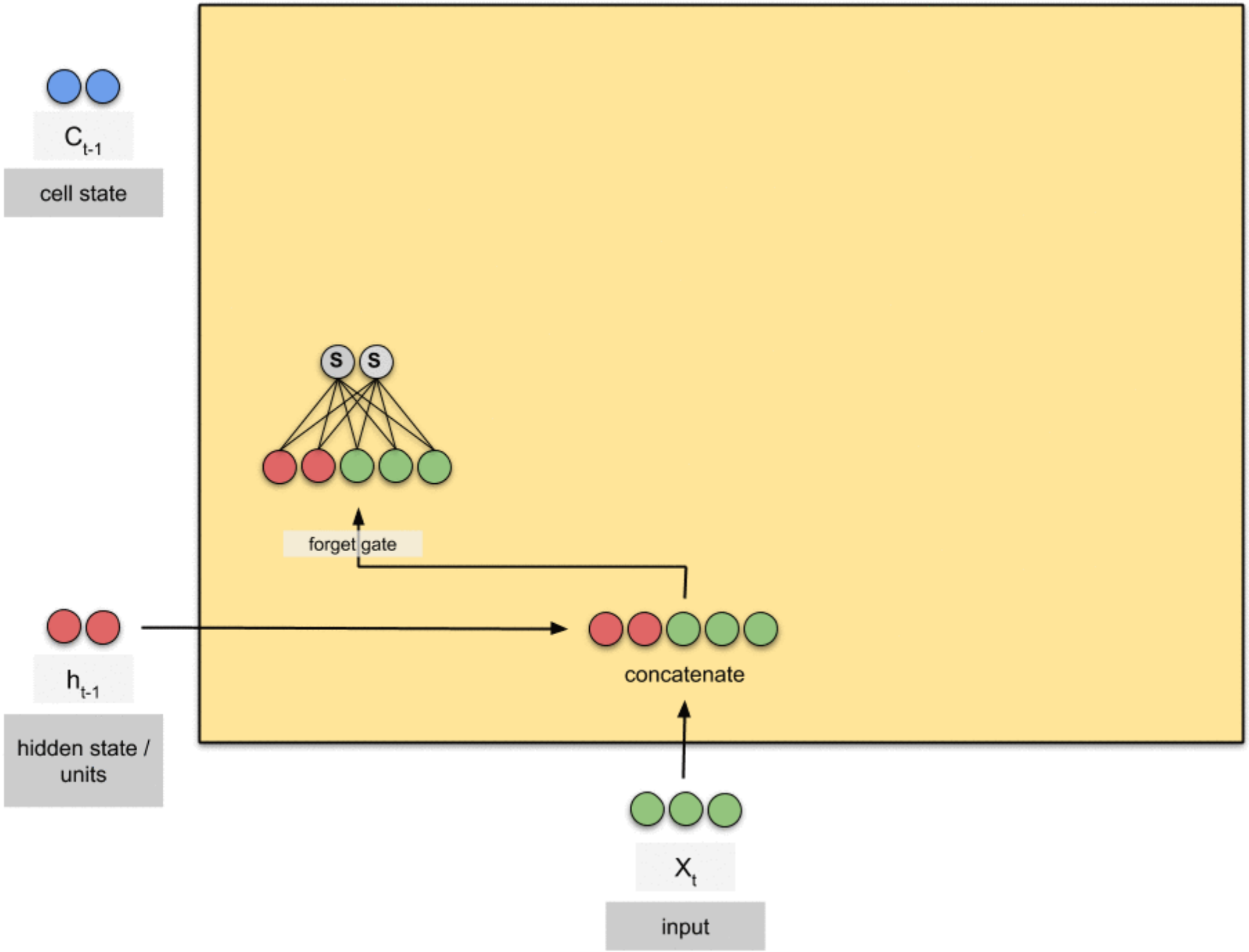
Christopher Olah's tutorial: Understanding LSTM Networks

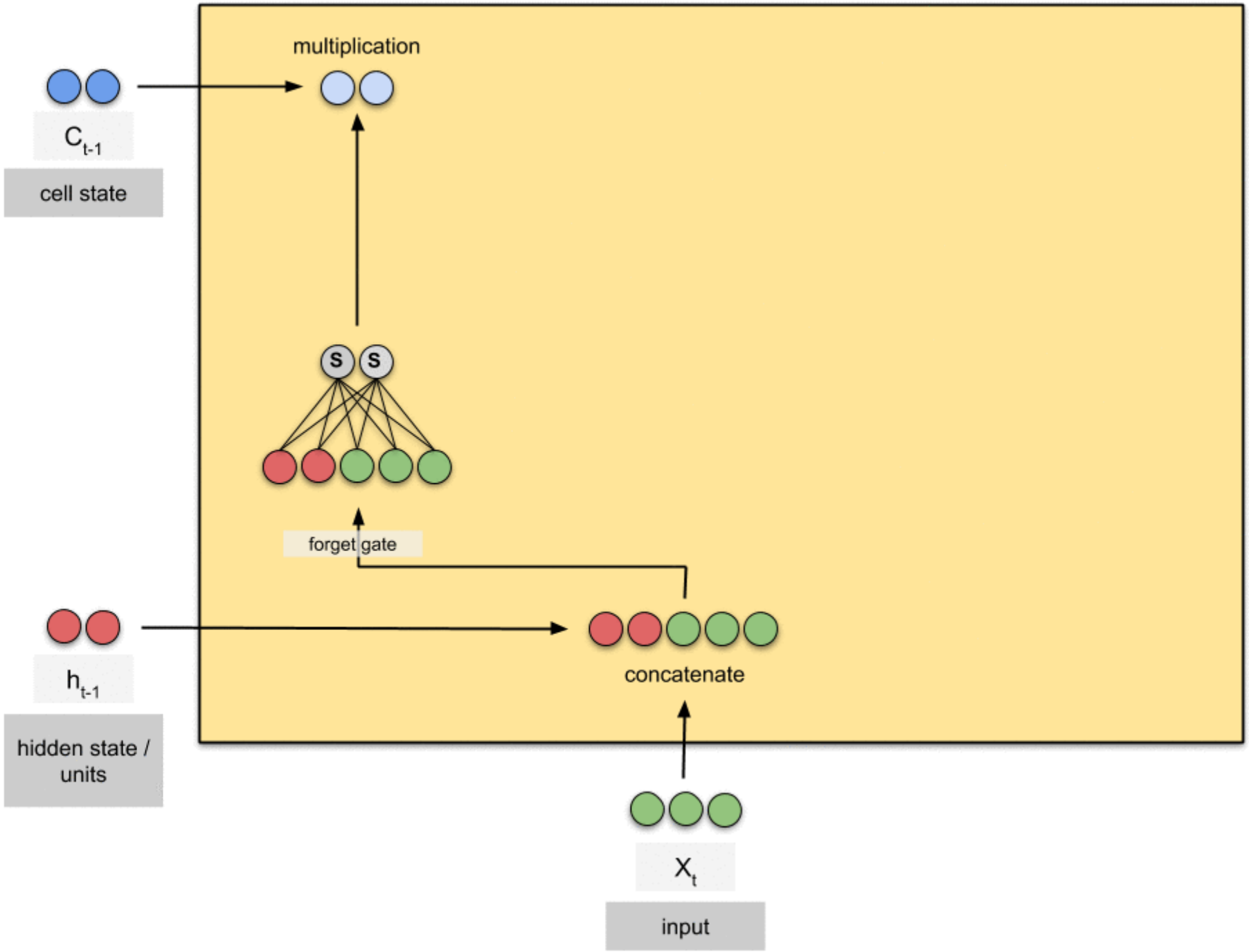
https://colah.github.io/posts/2015-08-Understanding-LSTMs

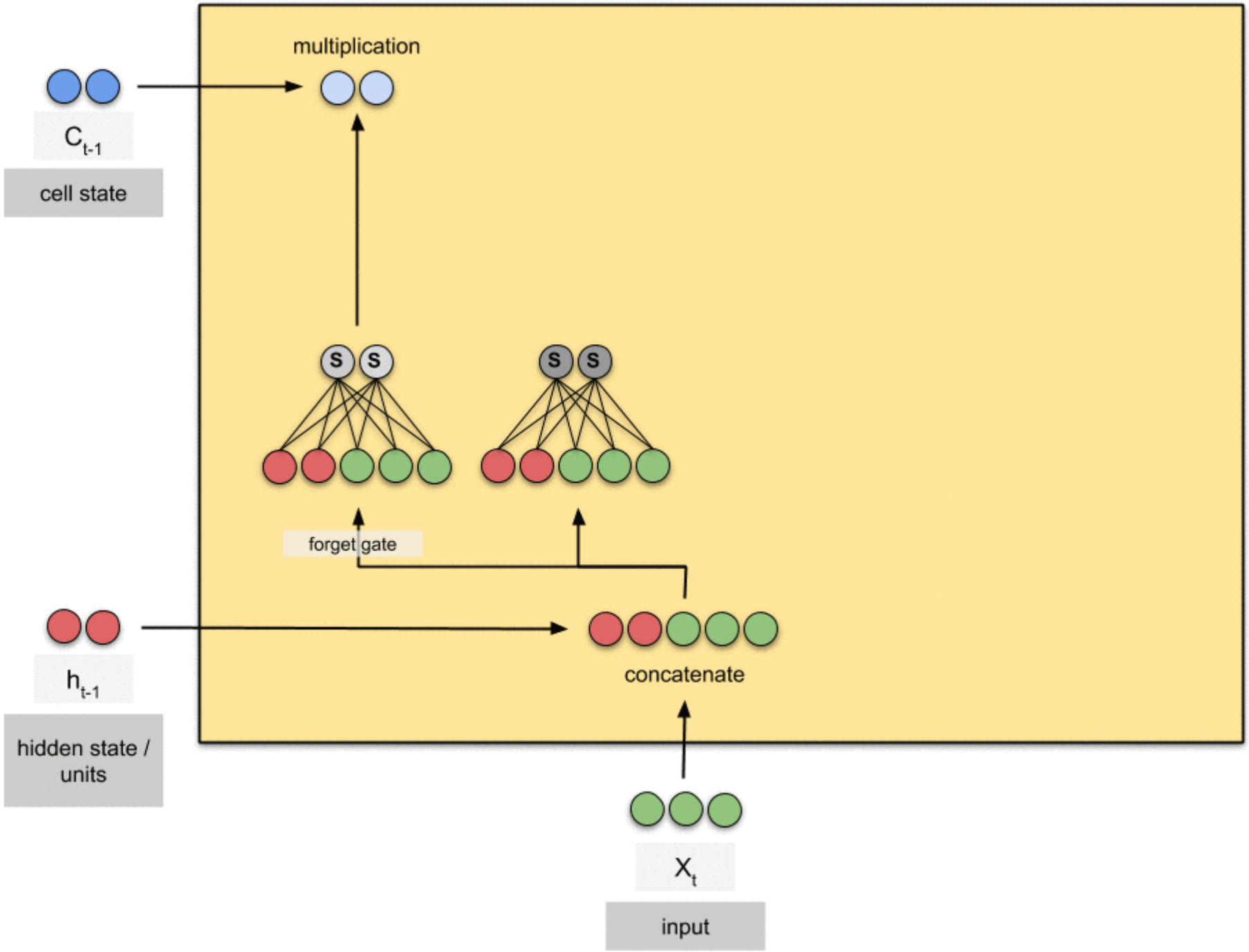

long
short-term
memory
prediction
Output gate
selection
collected
possibilities
forgetting
memory
filtered
possibilities
ignoring
Input gate
possibilities
new
information

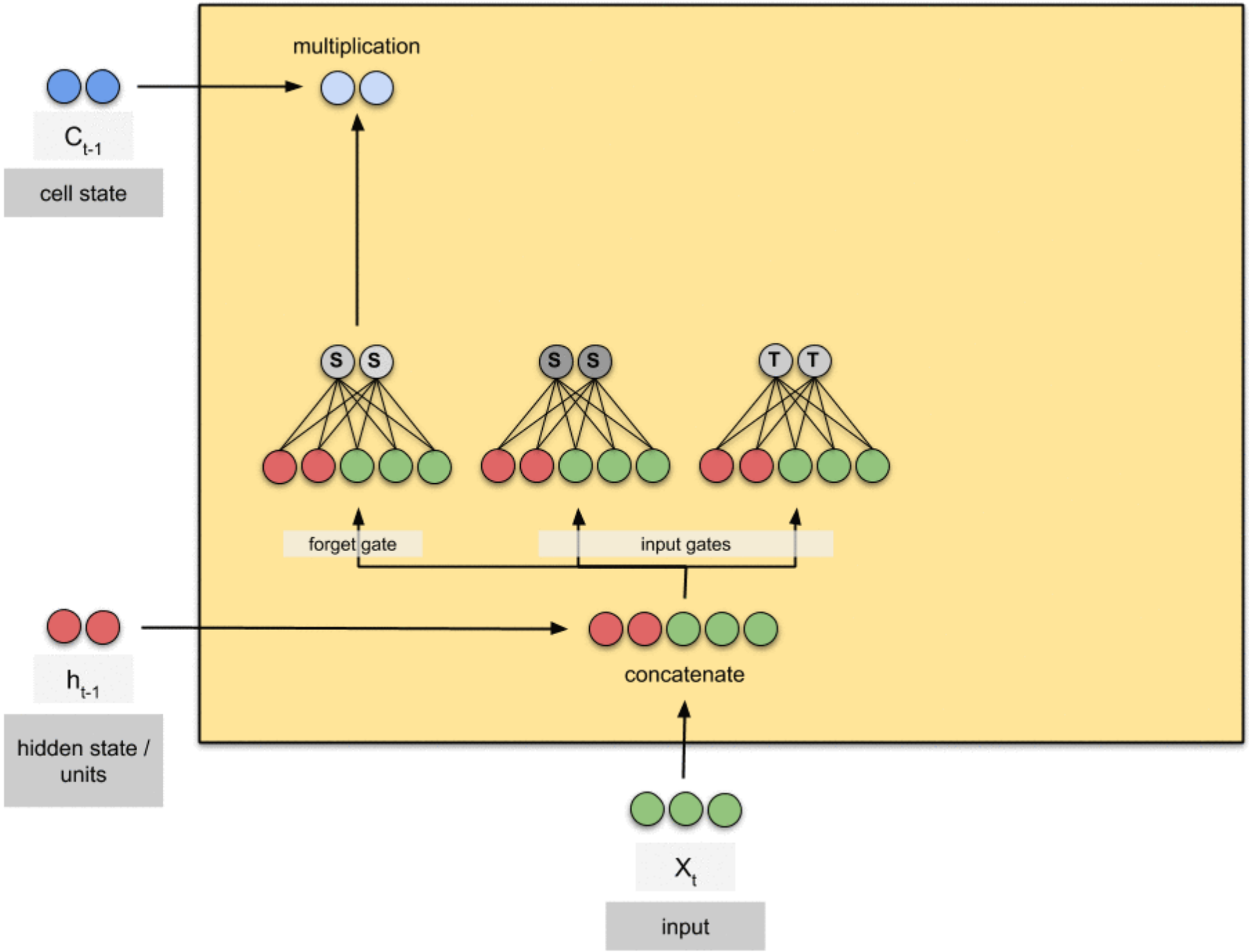

C_{t-1}
cell state
h_{t-1}
hidden state / units
X_t
input

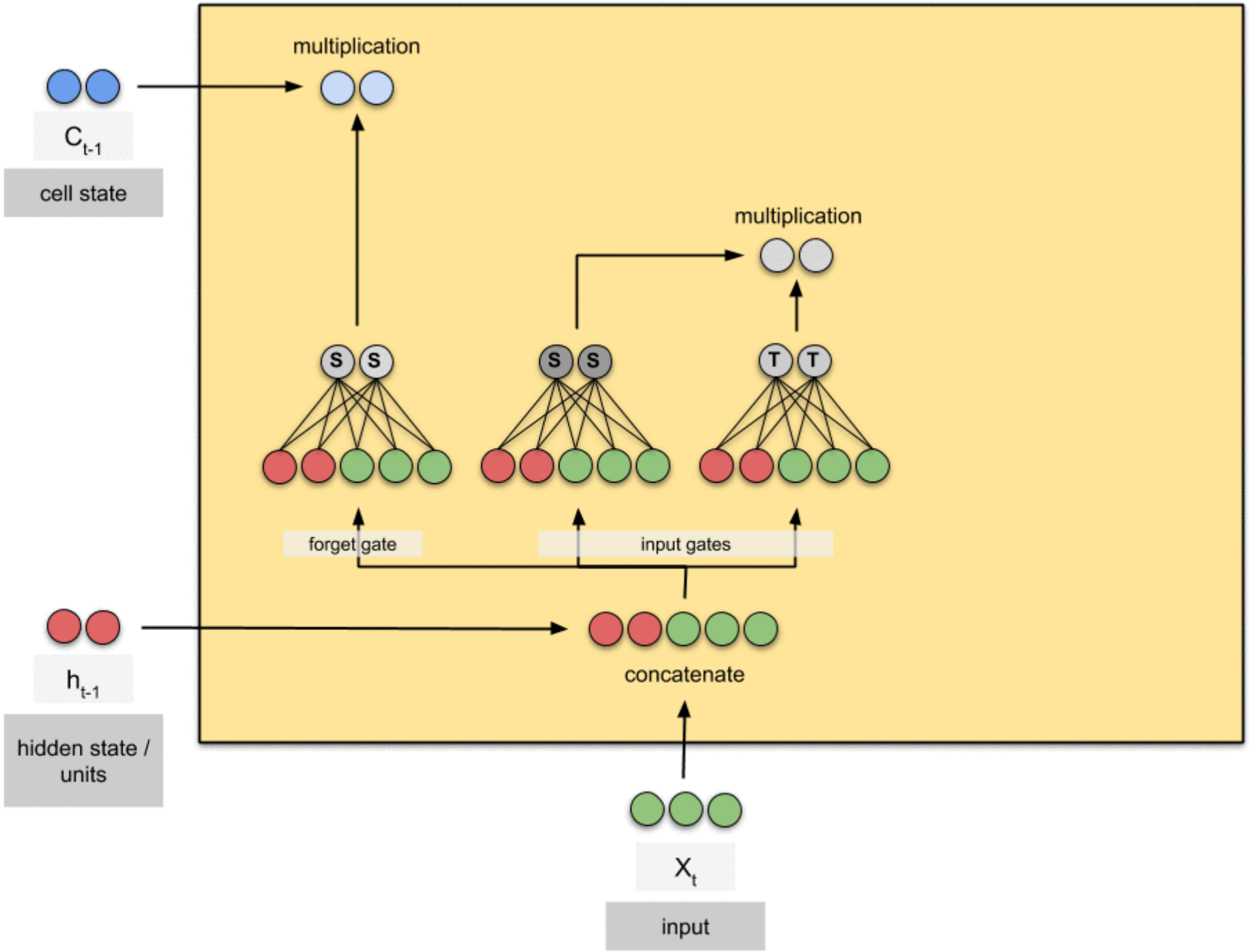

C_{t-1}
cell state
h_{t-1}
hidden state / units
concatenate
X_t
input

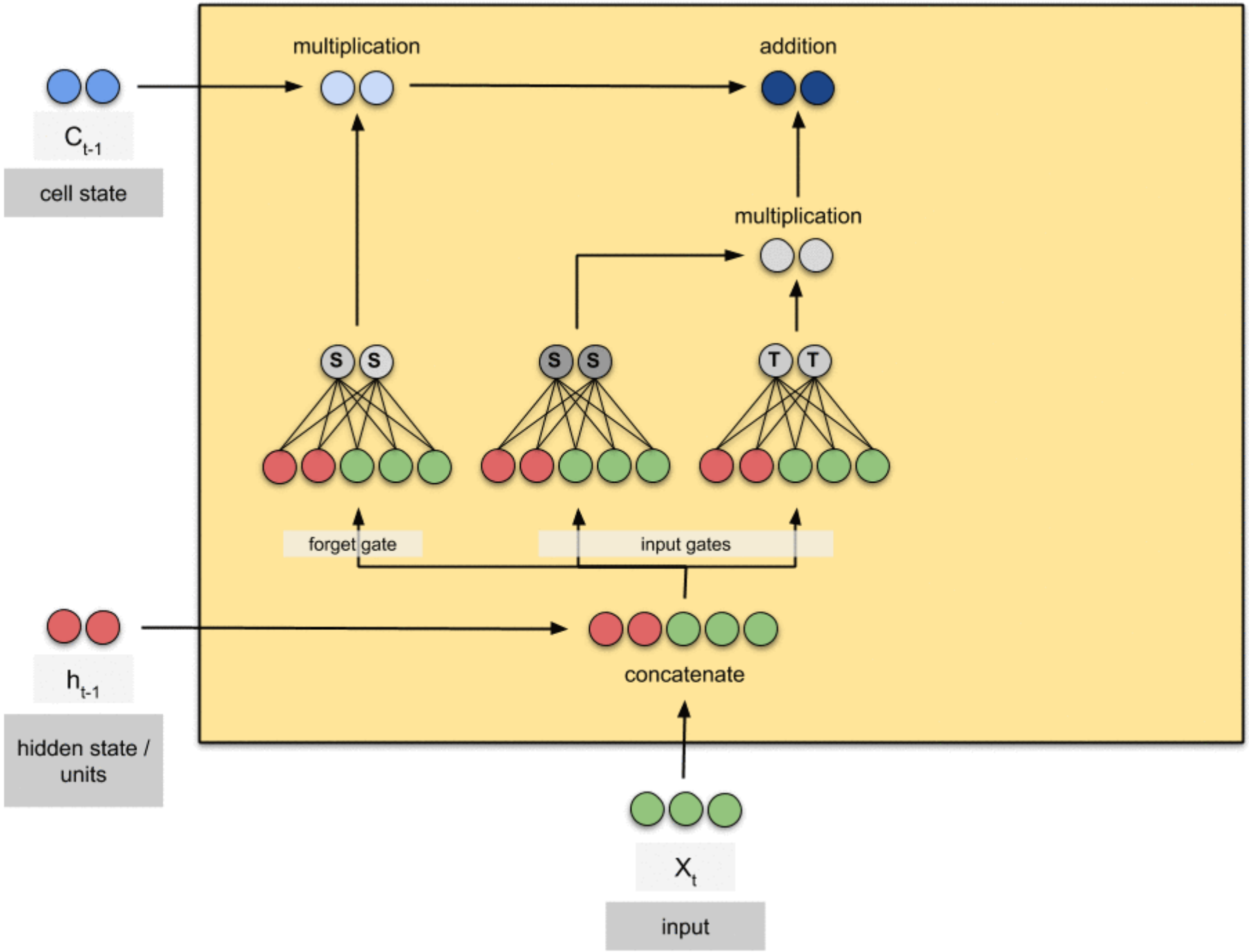

C_{t-1}
cell state
S
S
forget gate
h_{t-1}
hidden state / units
concatenate
X_t
input

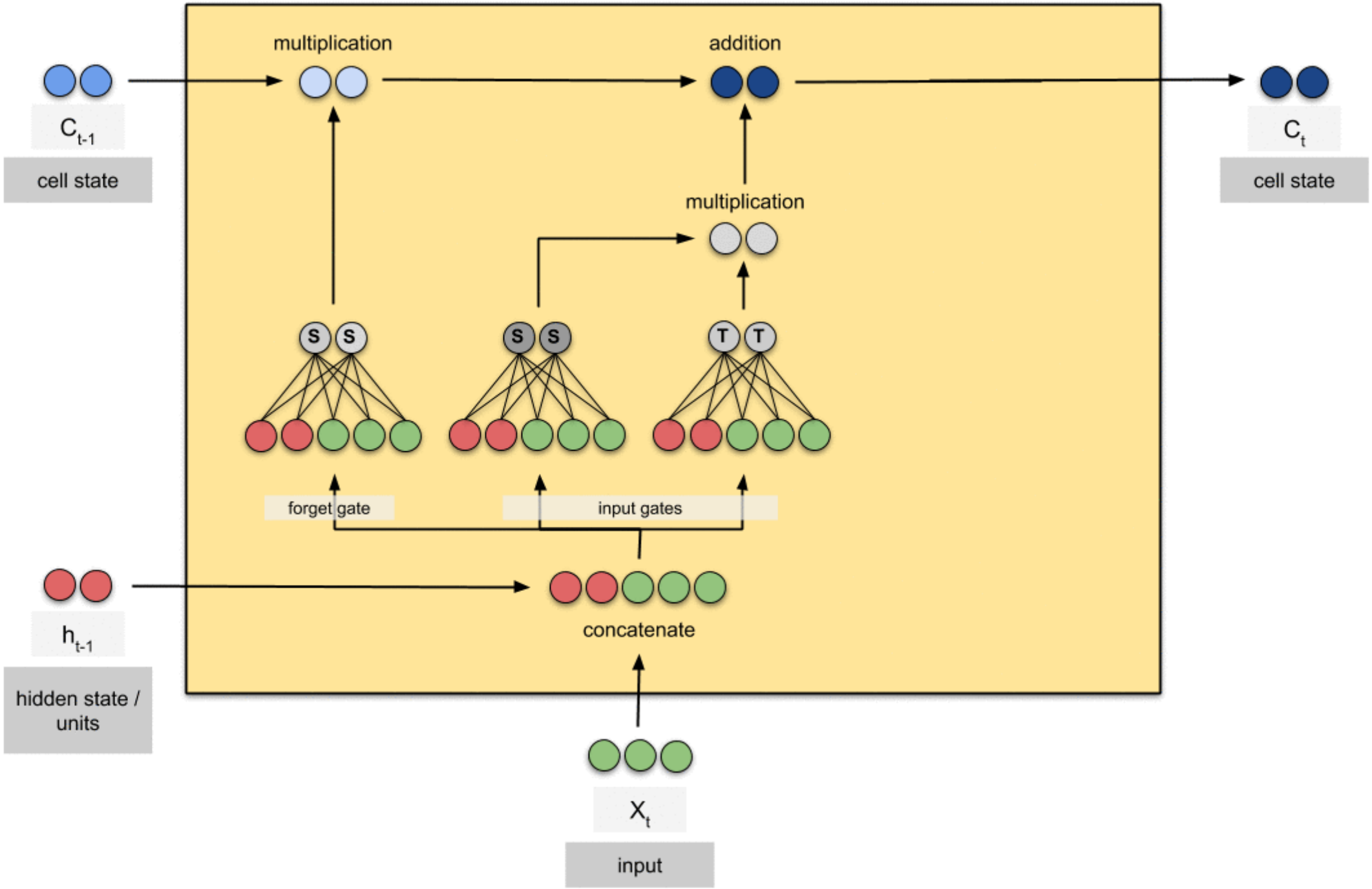

multiplication
C_{t-1}
cell state
S
S
forget gate
h_{t-1}
hidden state / units
concatenate
X_t
input

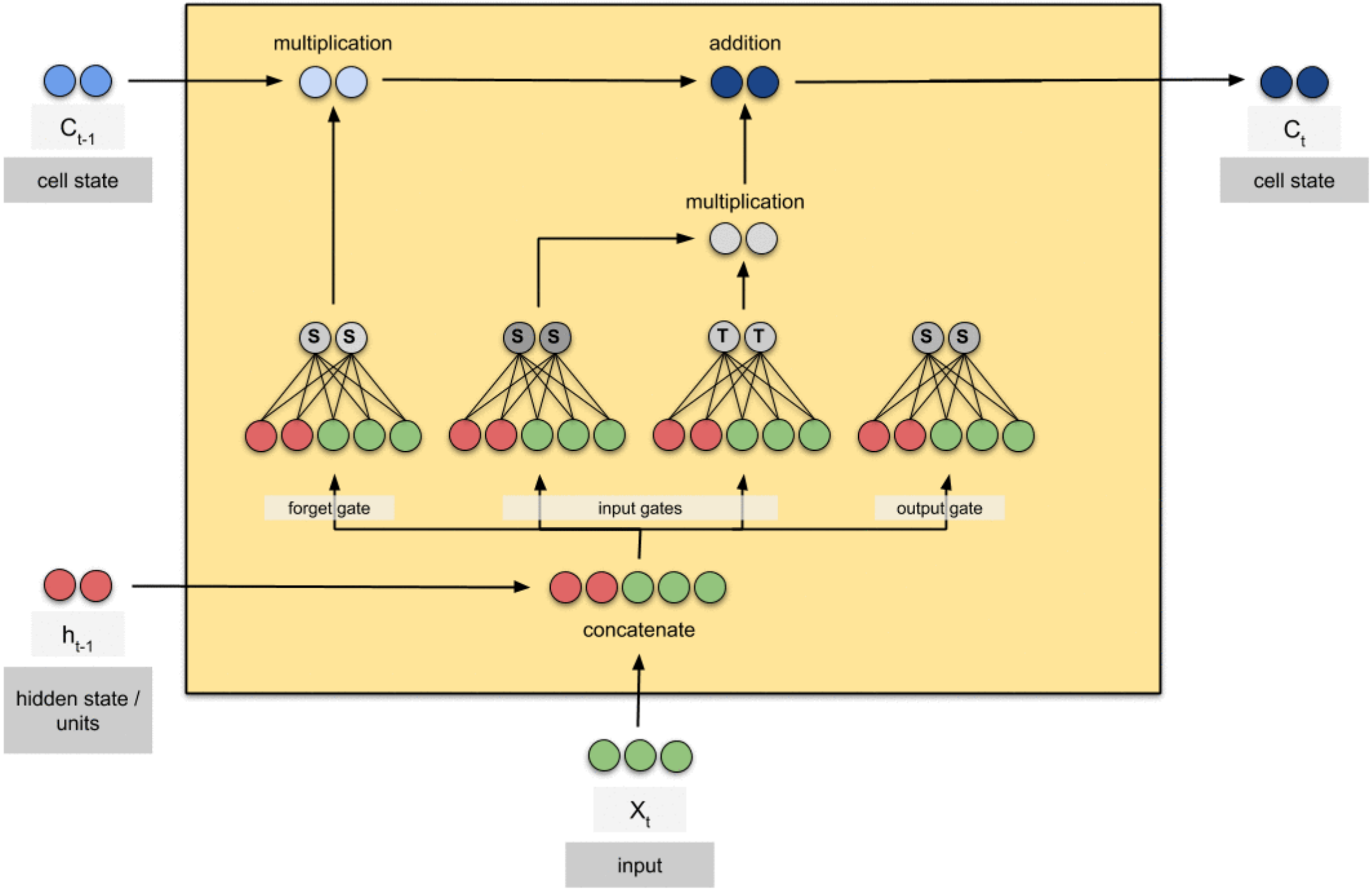

multiplication
C_{t-1}
cell state
S
S
S
S
forget gate
h_{t-1}
hidden state / units
concatenate
X_t
input

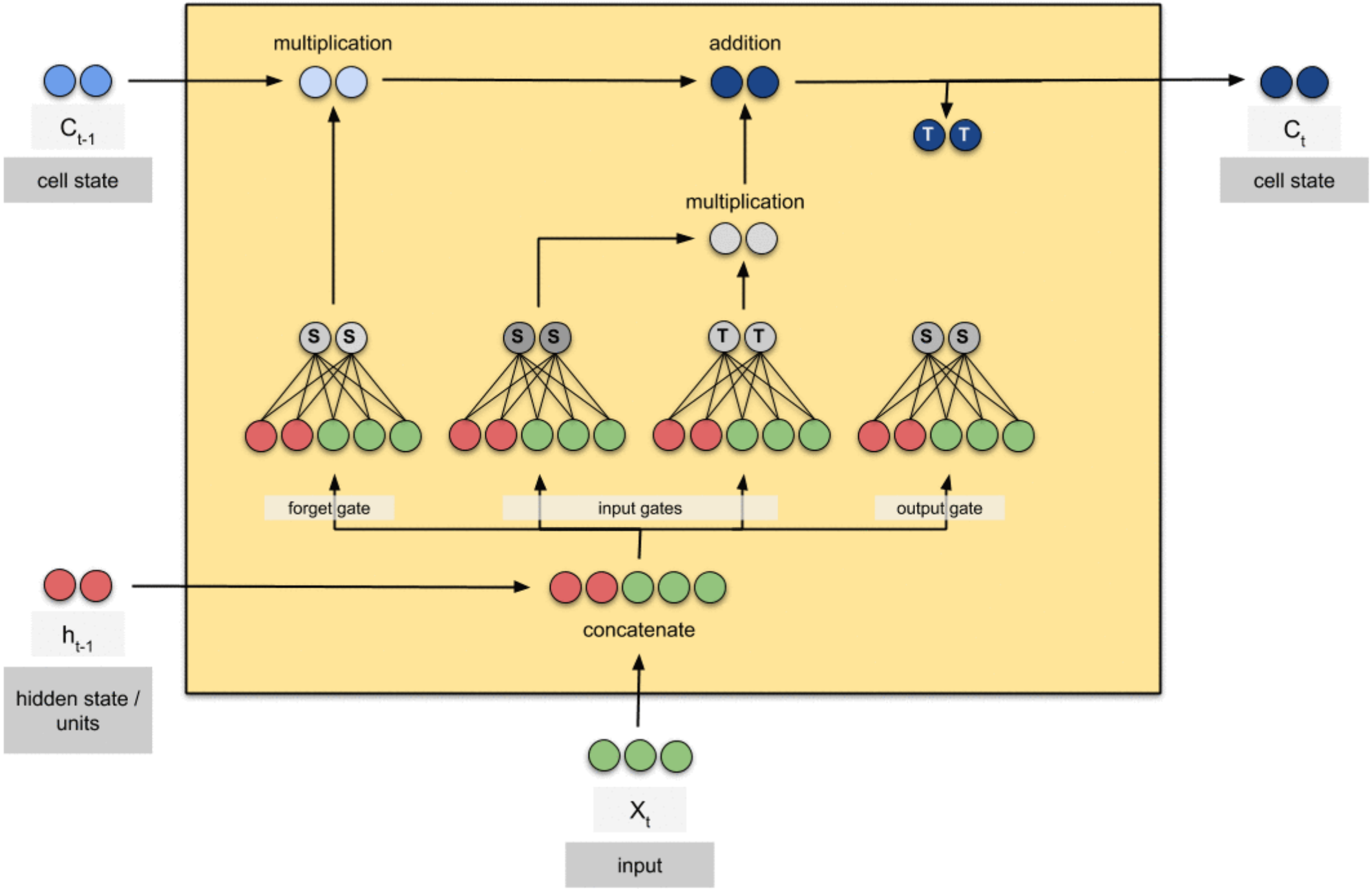

multiplication
C_{t-1}
cell state
S
S
S
S
T
T
forget gate
input gates
h_{t-1}
hidden state / units
concatenate
X_t
input

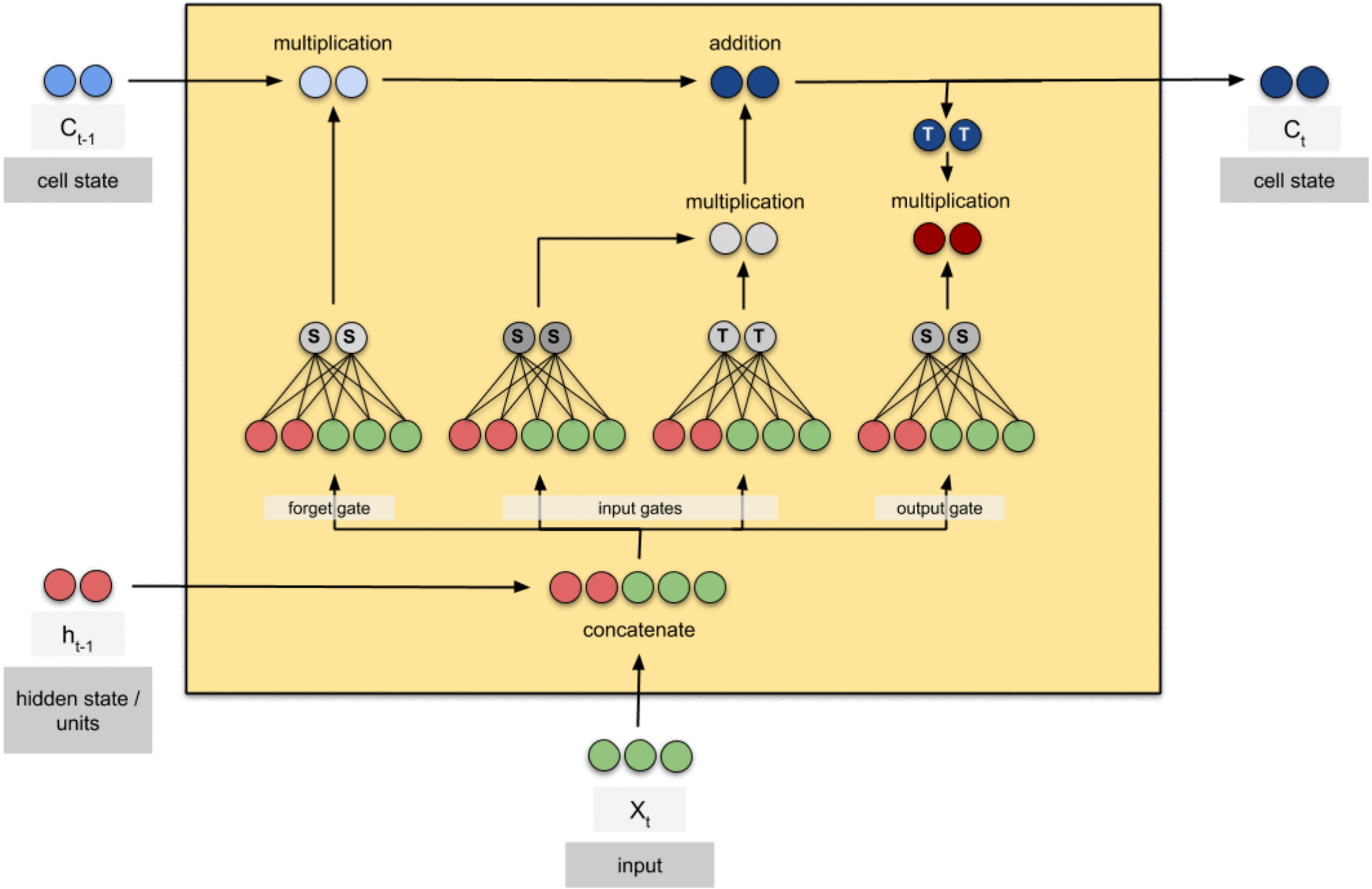

multiplication
C_{t-1}
cell state
multiplication
S
S
S
S
T
T
forget gate
input gates
h_{t-1}
hidden state / units
concatenate
X_t
input

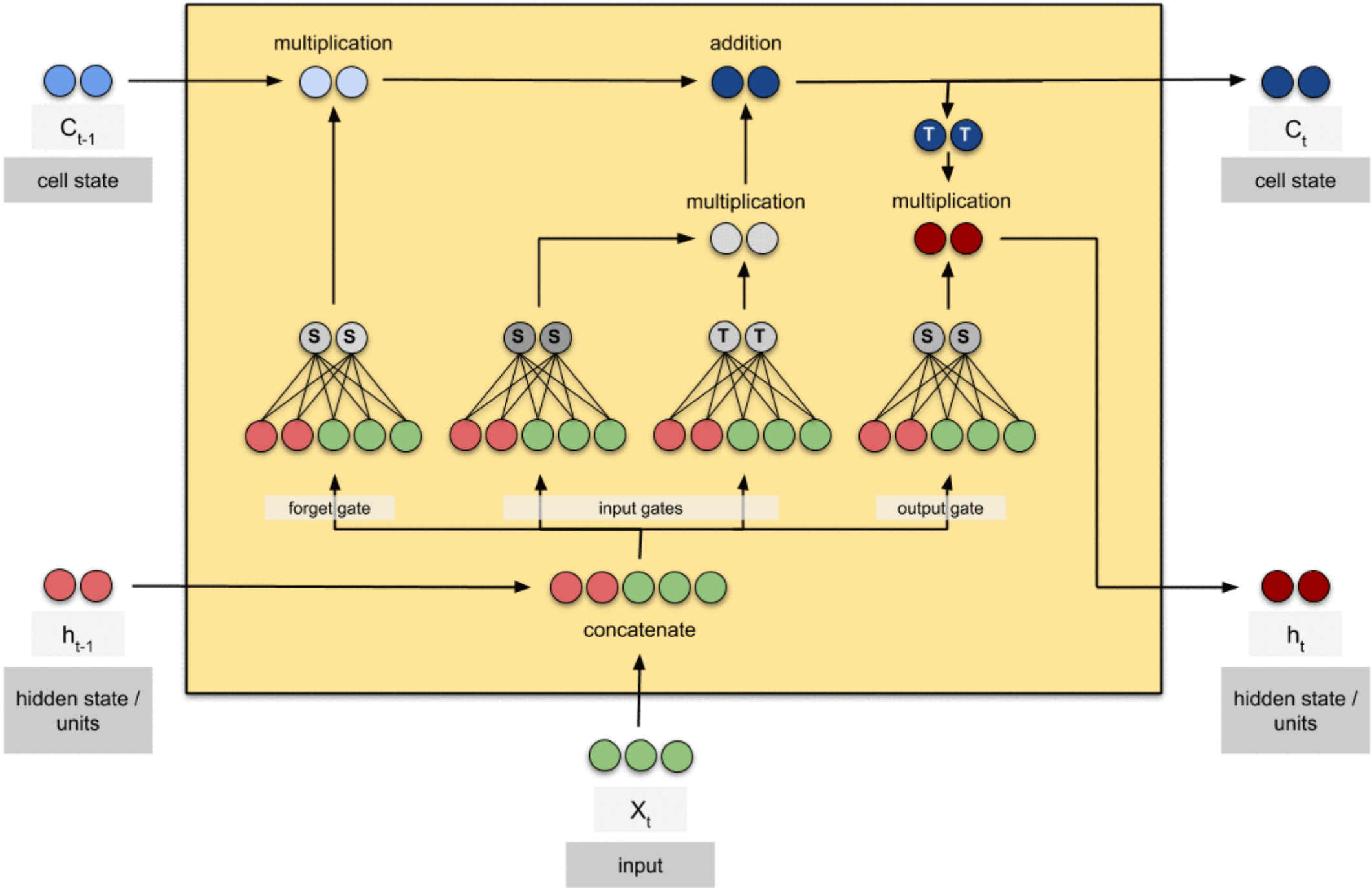

multiplication
addition
C_{t-1}
cell state
multiplication
S
S
S
S
T
T
forget gate
input gates
h_{t-1}
hidden state / units
concatenate
X_t
input


multiplication
addition
C_{t-1}
cell state
C_t
cell state
multiplication
S
S
S
S
T
T
forget gate
input gates
h_{t-1}
hidden state / units
concatenate
X_t
input


multiplication
addition
C_{t-1}
cell state
C_t
cell state
multiplication
S
S
S
S
T
T
S
S
forget gate
input gates
output gate
h_{t-1}
hidden state / units
concatenate
X_t
input

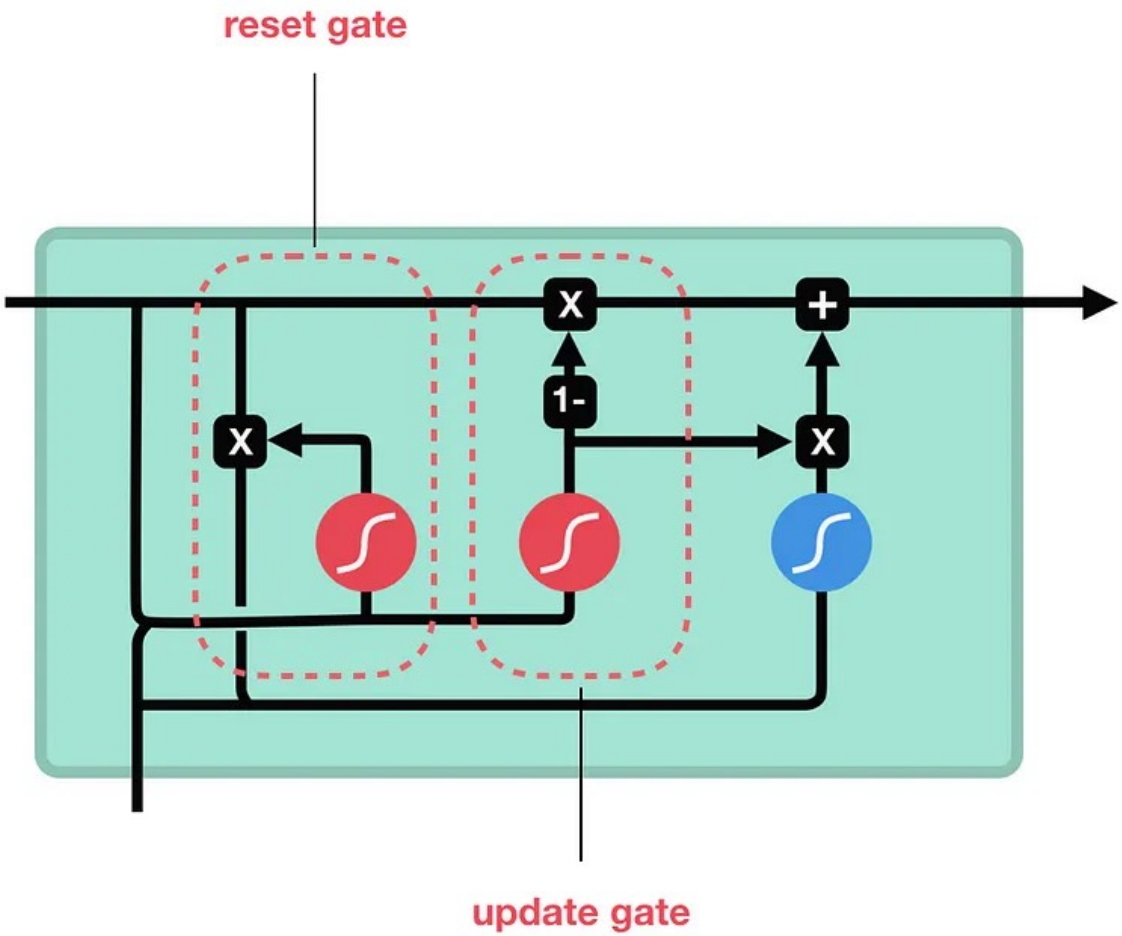

multiplication
addition
C_{t-1}
cell state
T T
C_t
cell state
multiplication
S S
S S
T T
S S
forget gate
input gates
output gate
h_{t-1}
hidden state / units
concatenate
X_t
input


multiplication
addition
C_{t-1}
cell state
C_t
cell state
T T
multiplication
multiplication
S S
S S
T T
S S
forget gate
input gates
output gate
h_{t-1}
hidden state / units
concatenate
X_t
input


multiplication
addition
C_{t-1}
cell state
C_t
cell state
T T
multiplication
multiplication
S S
S S
T T
S S
forget gate
input gates
output gate
h_{t-1}
hidden state / units
concatenate
h_t
hidden state / units
X_t
input

GRU (Gated Recurrent Unit)

Introduced in 2014 by Cho et al.
Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation
https://arxiv.org/pdf/1406.1078v3



·GRU combine the forget and input gates into a single “update gate.”

· It also merges the cell state and hidden state

· By simplifying the network structure they can become faster to train