Module 4

Processing Sequences using RNNs and CNNs

Introduction

Recurrent neural networks (RNNs) - a class of nets that can predict the future.

RNNs can analyze time series data, such as the number of daily active users on your website, the hourly temperature in your city, your home’s daily power consumption, and more.

RNNs can work on sequences of arbitrary lengths, rather than on fixed-sized inputs.

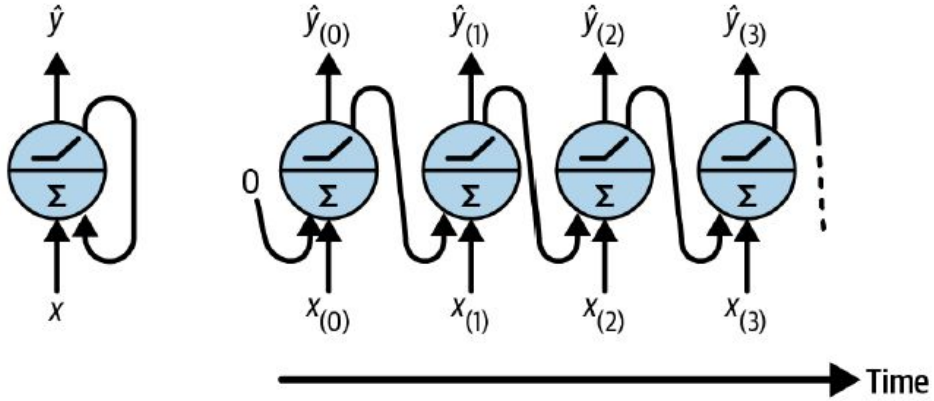
RNNs are not the only types of neural networks capable of handling sequential data.
For small sequences, a regular dense network can work, and for very long sequences, such as audio samples or text, convolutional neural networks can actually work quite well too.

Recurrent Neurons and Layers

Feedforward neural networks => the activations flow only in one direction, from the input layer to the output layer.

A recurrent neural network looks very much like a feedforward neural network, except it also has connections pointing backward.

Simplest RNN composed of one neuron receiving inputs, producing an output, and sending that output back to itself.

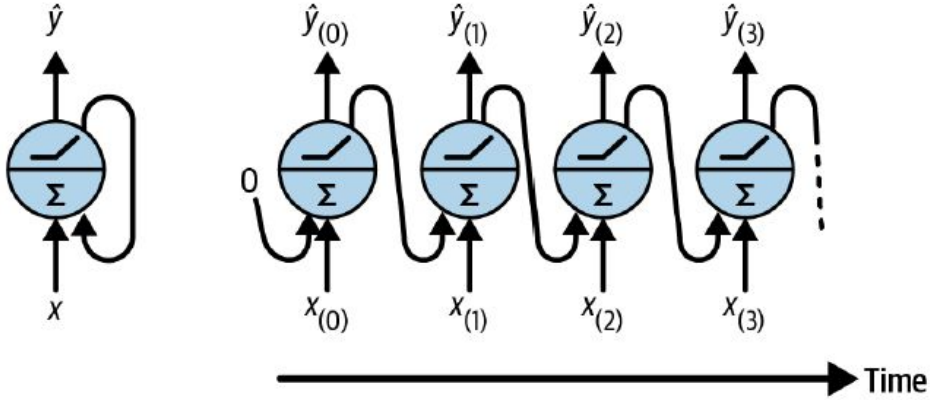


A recurrent neuron (left) unrolled through time (right)

Recurrent Neurons and Layers

At each time step t, this recurrent neuron receives the inputs x(t) as well as its own output from the previous time step, ŷ(t–1).

Since there is no previous output at the first time step, it is generally set to 0.

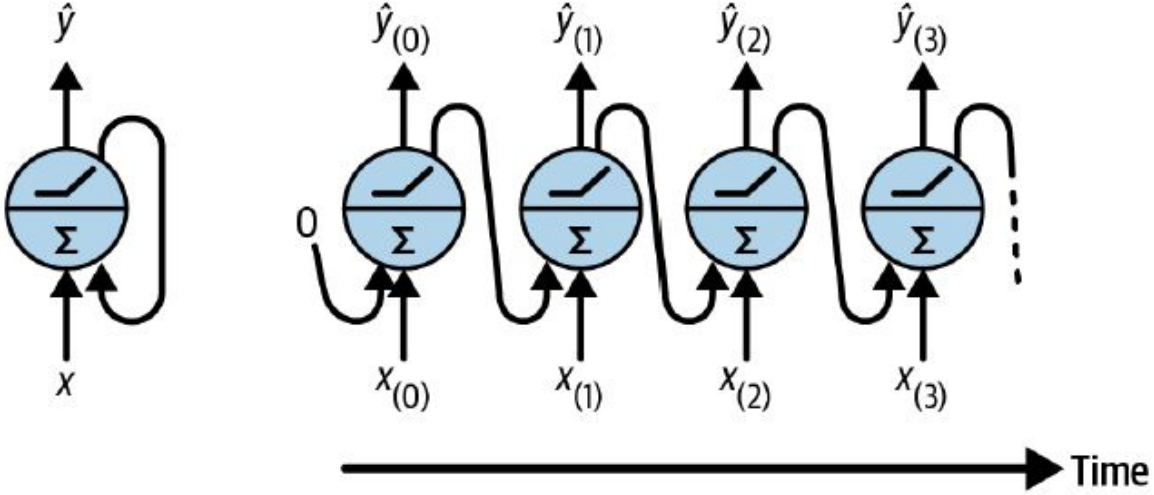


A recurrent neuron (left) unrolled through time (right)

Memory Cells

Since the output of a recurrent neuron at time step t is a function of all the inputs from previous time steps, you could say it has a form of memory.

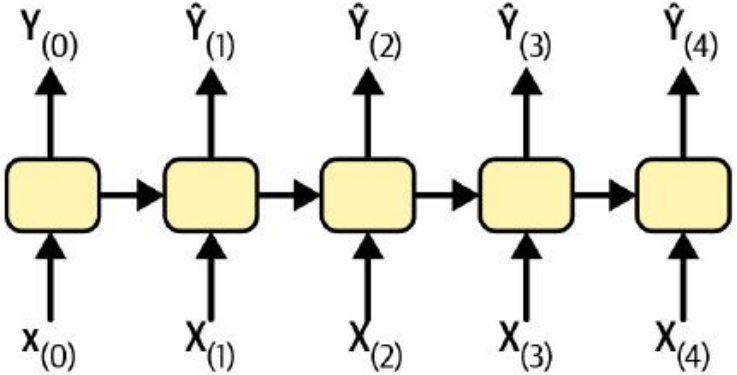
A part of a neural network that preserves some state across time steps is called a memory cell (or simply a cell).



Input and Output Sequences

An RNN can simultaneously take a sequence of inputs and produce a sequence of outputs.

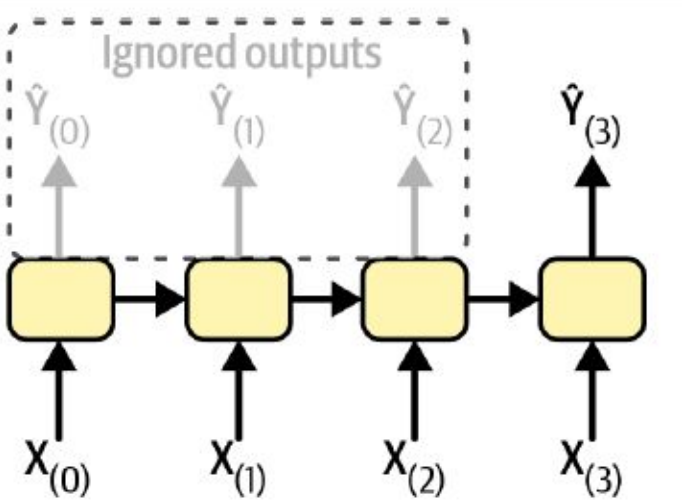
This type of sequence-to-sequence network is useful to forecast time series, such as your home's daily power consumption: you feed it the data over the last N days, and you train it to output the power consumption shifted by one day into the future (i.e., from N – 1 days ago to tomorrow).



Input and Output Sequences

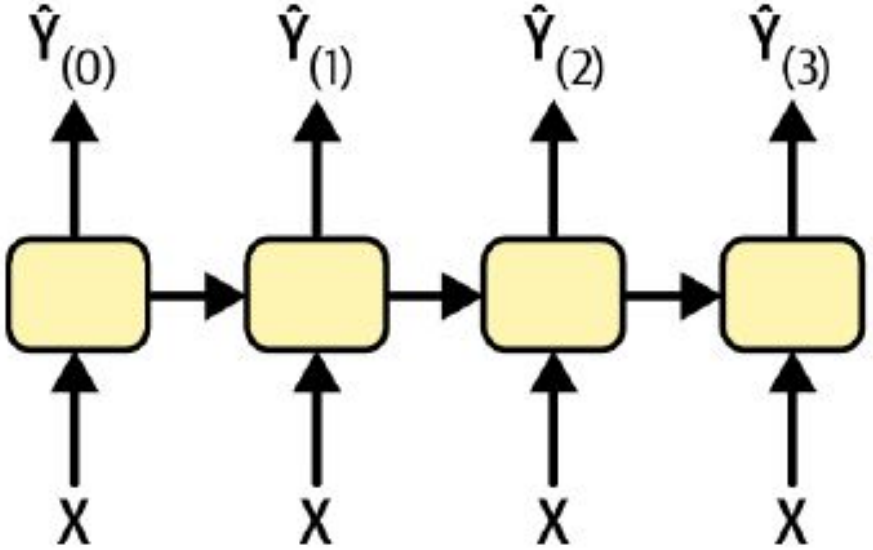
Alternatively, you could feed the network a sequence of inputs and ignore all outputs except for the last one. This is a sequence-to-vector network.

For example, you could feed the network a sequence of words corresponding to a movie review, and the network would output a sentiment score (e.g., from 0 [hate] to 1 [love]).



Input and Output Sequences

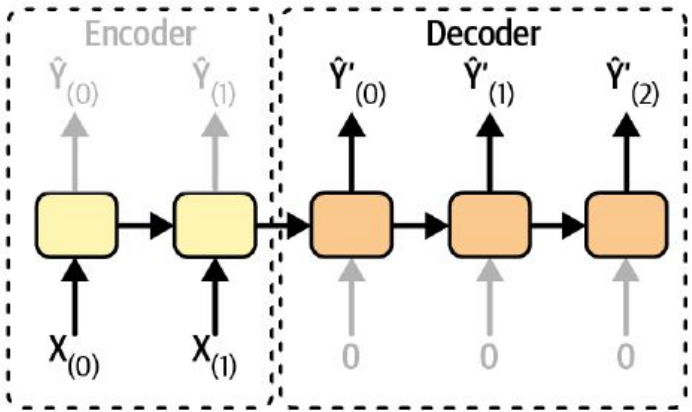
You could feed the network the same input vector over and over again at each time step and let it output a sequence. This is a vector-to-sequence network. For example, the input could be an image, and the output could be a caption for that image.



Input and Output Sequences

You could have a sequence-to-vector network, called an encoder, followed by a vector-to-sequence network, called a decoder.

For example, this could be used for translating a sentence from one language to another. You would feed the network a sentence in one language, the encoder would convert this sentence into a single vector representation, and then the decoder would decode this vector into a sentence in another language.



This two-step model, called an encoder-decoder, works much better than trying to translate on the fly with a single sequence-to-sequence RNN: the last words of a sentence can affect the first words of the translation, so you need to wait until you have seen the whole sentence before translating it.

Training RNNs

To train an RNN, the trick is to unroll it through time and then use regular backpropagation. This strategy is called backpropagation through time (BPTT).

Simple RNNs can be quite good at forecasting time series or handling other kinds of sequences, but they do not perform as well on long time series or sequences, because when an RNN processes a long sequence, it will gradually forget the first inputs in the sequence.

Just like any deep neural network it may suffer from the unstable gradients problem.

It may take forever to train, or training may be unstable.

Fighting the Unstable Gradients Problem

Many of the tricks we used in deep nets to alleviate the unstable gradients problem can also be used for RNNs: good parameter initialization, faster optimizers, dropout, and so on.

Tackling the Short-Term Memory Problem

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.

Imagine you are trying to translate a long sentence; by the time you finished reading it, you have no clue how it started.

To tackle this problem, various types of cells with long-term memory have been introduced.

They have proven so successful that the basic cells are not used much anymore.

LSTM Cells

The long short-term memory (LSTM) cell was proposed in 1997 and gradually improved over the years.

If you consider the LSTM cell as a black box, it can be used very much like a basic cell, except it will perform much better; training will converge faster, and it will detect longer-term patterns in the data.

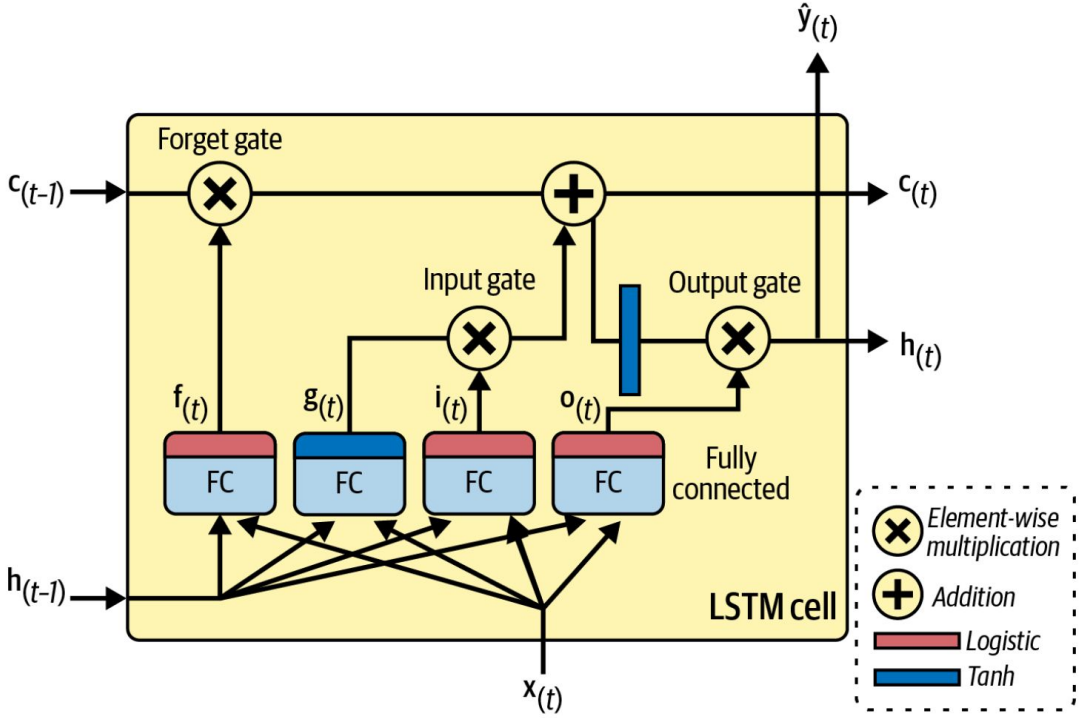
In Keras, you can simply use the LSTM layer instead of the SimpleRNN layer:

```
model = tf.keras.Sequential([
 tf.keras.layers.LSTM(32, return_sequences=True, input_shape=[None, 5]),
tf.keras.layers.Dense(14)
])
```

How does an LSTM cell work?

The LSTM cell looks exactly like a regular cell, except that its state is split into two vectors: **h**(*t*) and **c**(*t*) (“c” stands for “cell”). You can think of **h**(*t*) as the short-term state and **c**(*t*) as the long-term state.

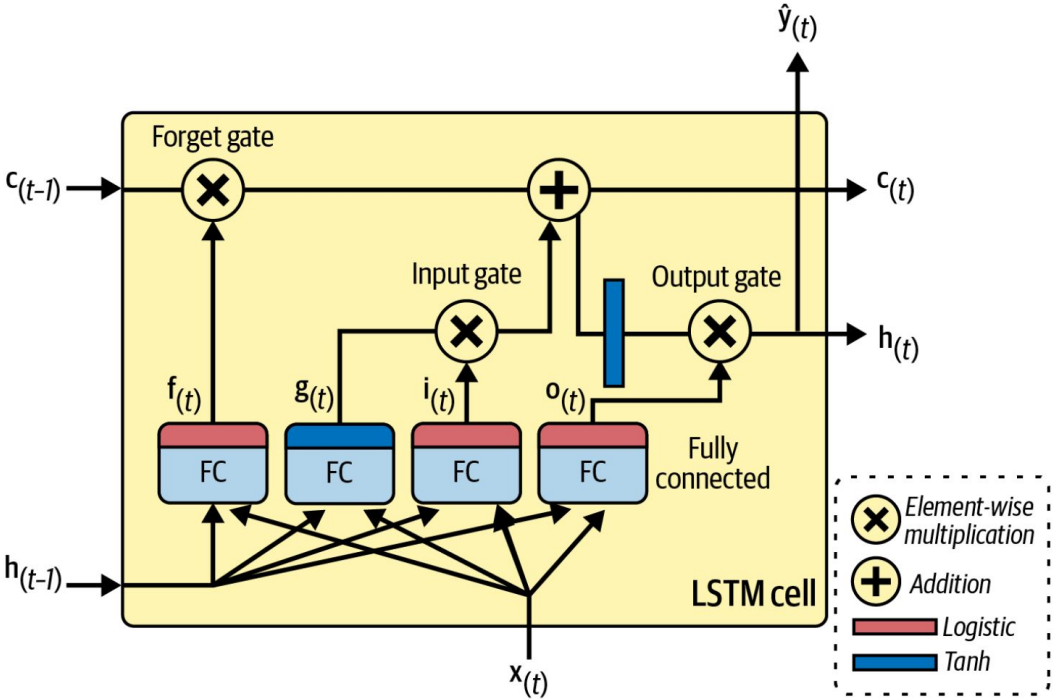
The key idea is that the network can learn what to store in the long-term state, what to throw away, and what to read from it.



How does an LSTM cell work?

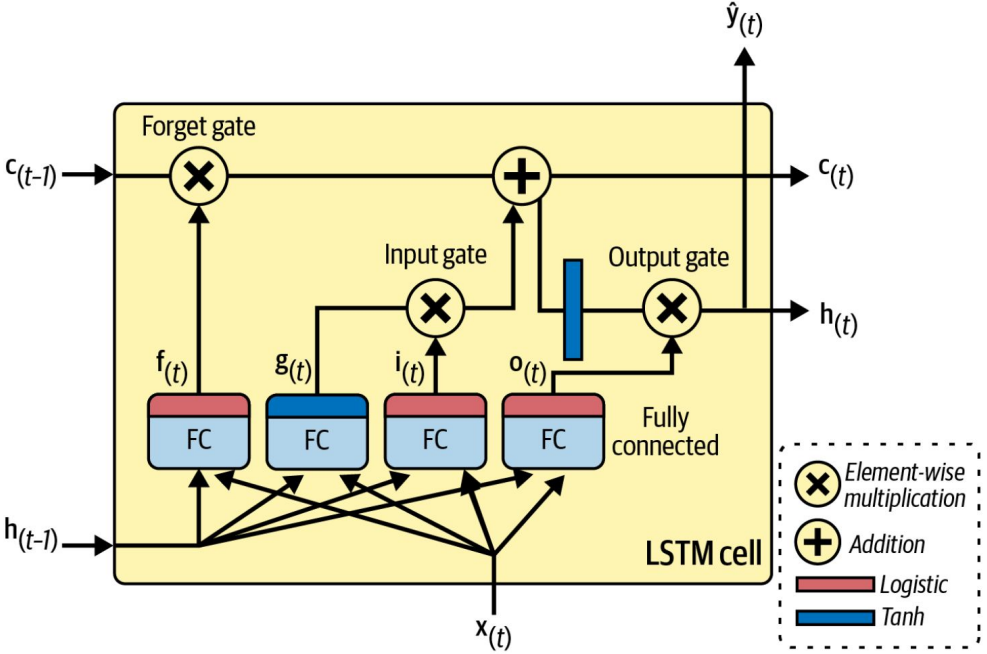
As the long-term state **c**(*t*–1) traverses the network from left to right, you can see that it first goes through a *forget gate*, dropping some memories, and then it adds some new memories via the addition operation (which adds the memories that were selected by an *input gate*).

The result **c**(*t*) is sent straight out, without any further transformation. So, at each time step, some memories are dropped and some memories are added.



How does an LSTM cell work?

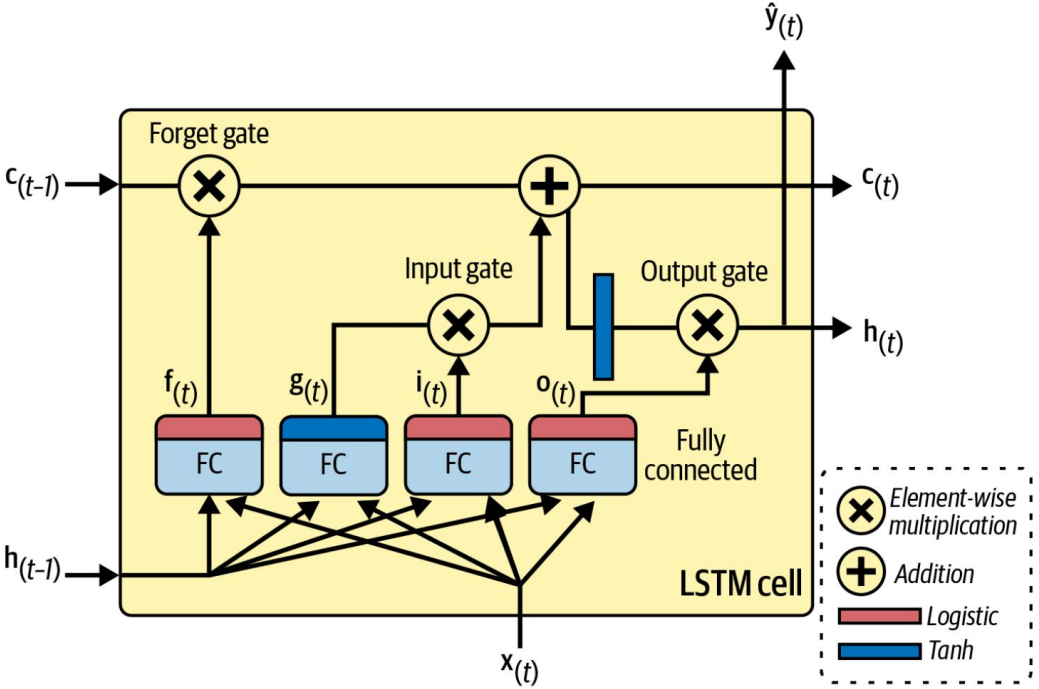
Moreover, after the addition operation, the long-term state is copied and passed through the tanh function, and then the result is filtered by the *output gate*. This produces the short-term state **h**(*t*) (which is equal to the cell's output for this time step, **y**(*t*)).



How does an LSTM cell work?

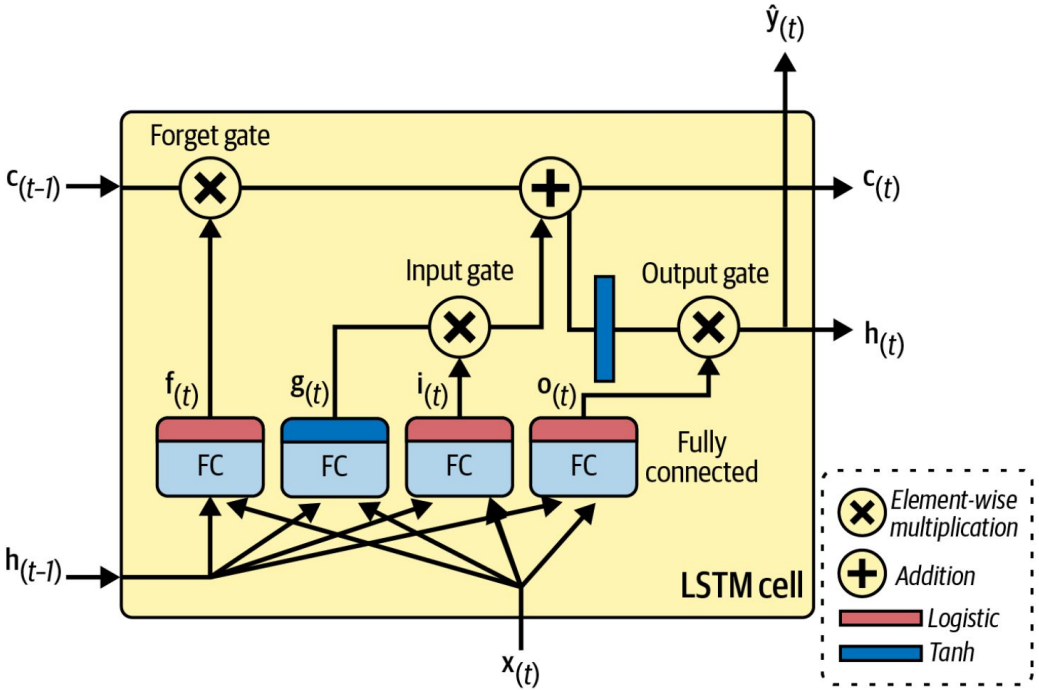
First, the current input vector **x**(*t*) and the previous short-term state **h**(*t*–1) are fed to four different fully connected layers.

They all serve a different purpose:



How does an LSTM cell work?

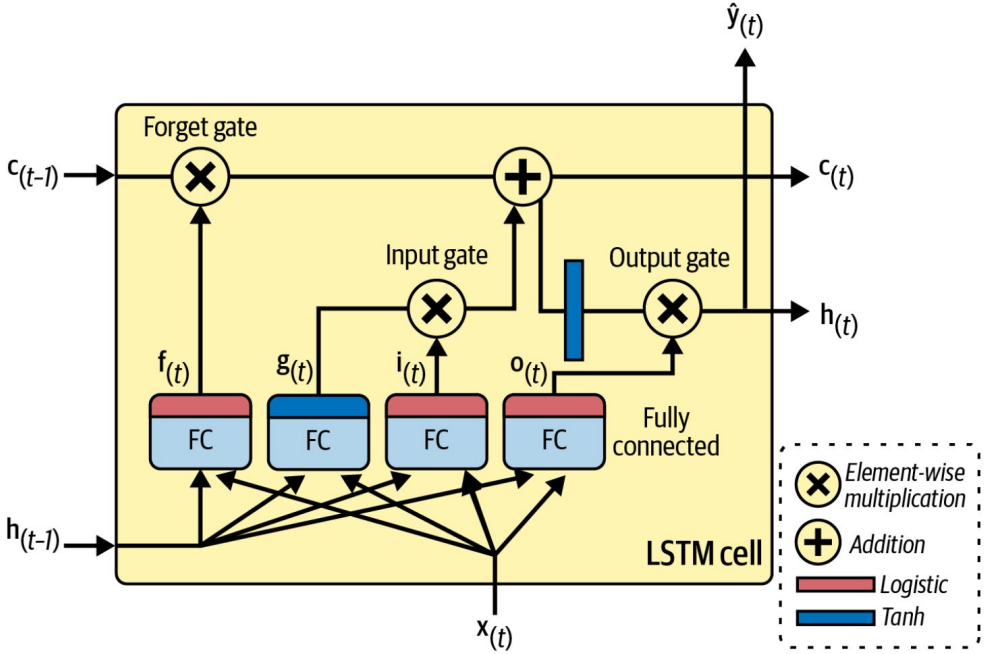
The main layer is the one that outputs g(t). It has the usual role of analyzing the current inputs x(t) and the previous (short-term) state h(t–1).



How does an LSTM cell work?

The three other layers are gate controllers. Since they use the logistic activation function, the outputs range from 0 to 1. The gate controllers' outputs are fed to element-wise multiplication operations: if they output 0s they close the gate, and if they output 1s they open it.

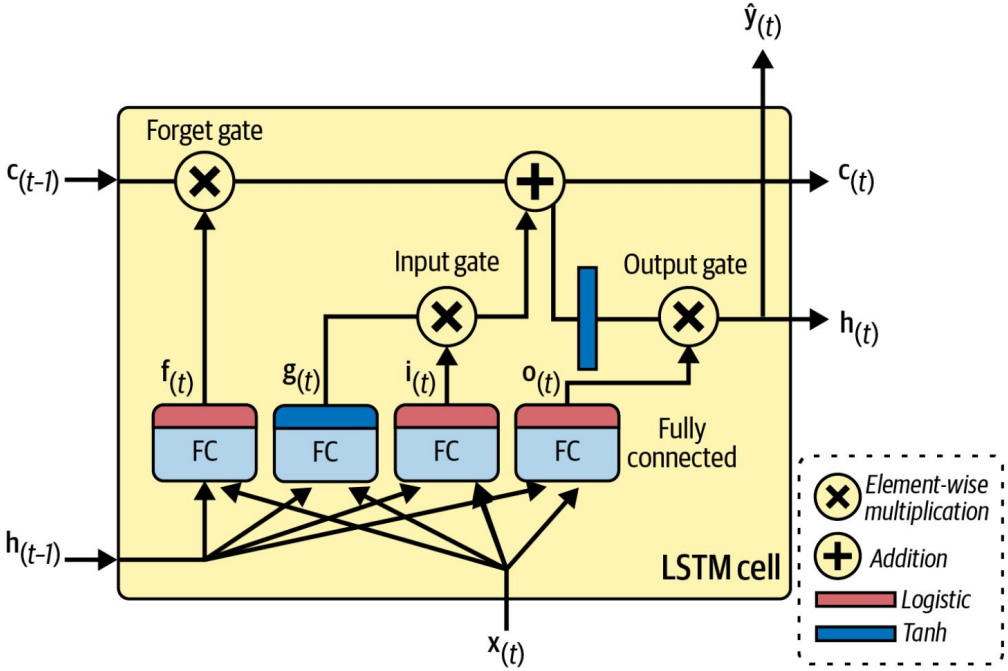
- The forget gate (controlled by f(t)) controls which parts of the long-term state should be erased.
- The input gate (controlled by i(t)) controls which parts of g(t) should be added to the long-term state.
- Finally, the output gate (controlled by o(t)) controls which parts of the long- term state should be read and output at this time step, both to h(t) and to y(t).



How does an LSTM cell work?

In short, an LSTM cell can learn to recognize an important input (that's the role of the input gate), store it in the long-term state, preserve it for as long as it is needed, and extract it whenever it is needed.

This explains why these cells have been amazingly successful at capturing long-term patterns in time series, long texts, audio recordings, and more.



LSTM Cells

LSTM cells are one of the main reasons behind the success of RNNs.

Yet while they can tackle much longer sequences than simple RNNs, they still have a fairly limited short-term memory, and they have a hard time learning long-term patterns in sequences of 100 time steps or more, such as audio samples, long time series, or long sentences.