Pontificia Universidad JAVERIANA Colombia

# Procesamiento de Lenguaje Natural

## Tópicos Avanzados en Analítica

## Maestría en Analítica para la Inteligencia de Negocios

Sergio Alberto Mora Pardo - H2 2023

# Recurrent Neural Network
# RNN, LSTM, GRU

Pontificia Universidad
JAVERIANA
Colombia

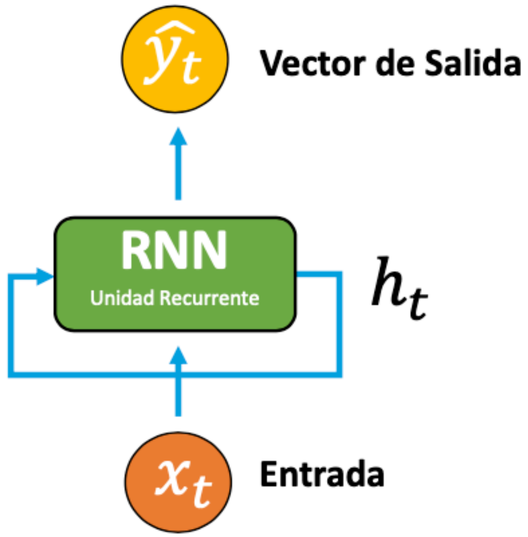
# RNN, LSTM y GRU

## Recurrent Neural Network

### Recurrent Neural Network

*Recibe entradas, produce una salida y envía esa salida a sí misma.*

### Recurrent Neurons and Layers




1. *Arquitectura que persiste los datos.*
2. *Modela dependencias a corto plazo*
3. *Manipula secuencias de longitud variable*
4. *Mantiene información sobre el orden*

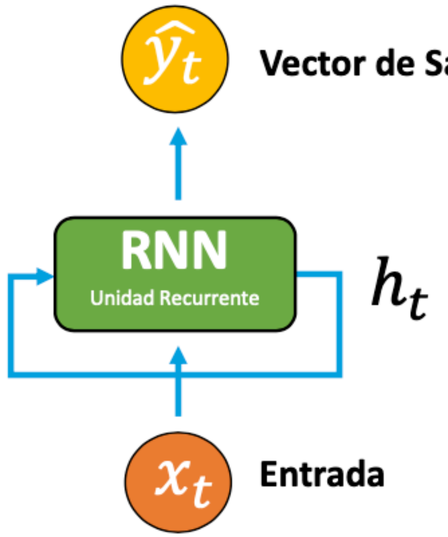
# RNN, LSTM y GRU

## Recurrent Neural Network

### Recurrent Neural Network

*Recibe entradas, produce una salida y envía esa salida a sí misma.*

### Recurrent Neurons and Layers




Estado de la Unidad — Función con parámetros W — Vector de entrada en T

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

Estado anterior

# RNN, LSTM y GRU

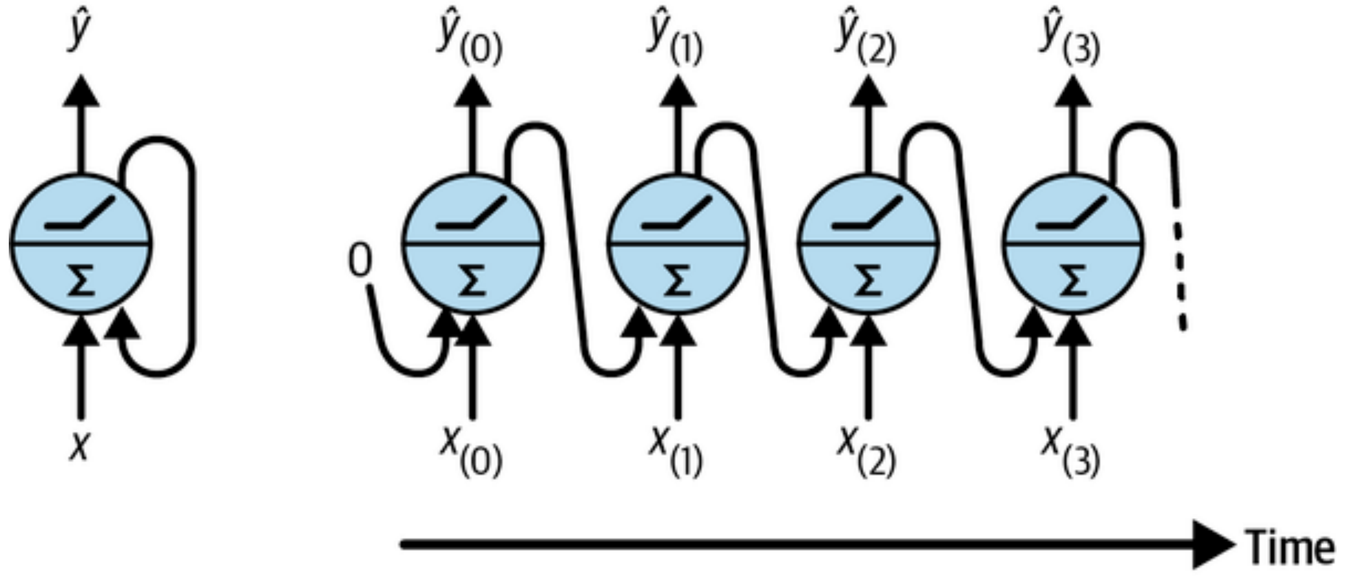
## Recurrent Neural Network

### Recurrent Neural Network

*Recibe entradas, produce una salida y envía esa salida a sí misma.*

### Recurrent Neurons and Layers

unrolling the network through



# RNN, LSTM y GRU

## Recurrent Neural Network

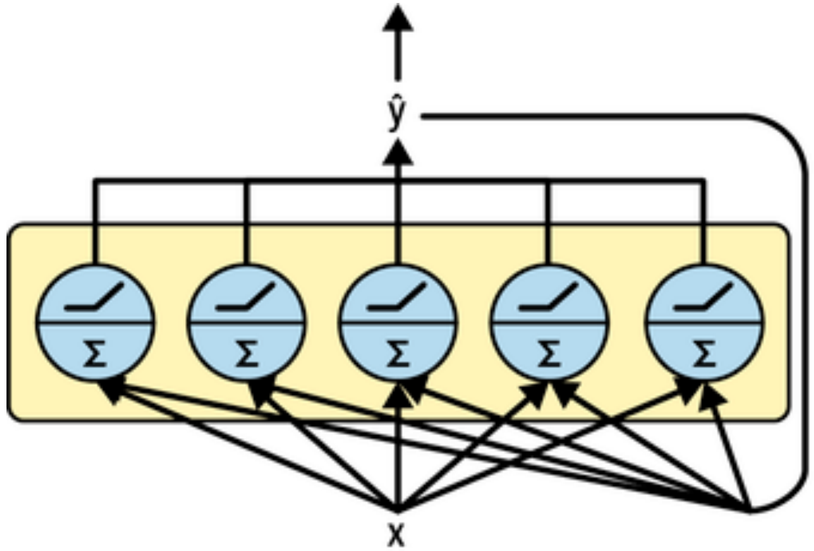
### Recurrent Neural Network

*Recibe entradas, produce una salida y envía esa salida a sí misma.*

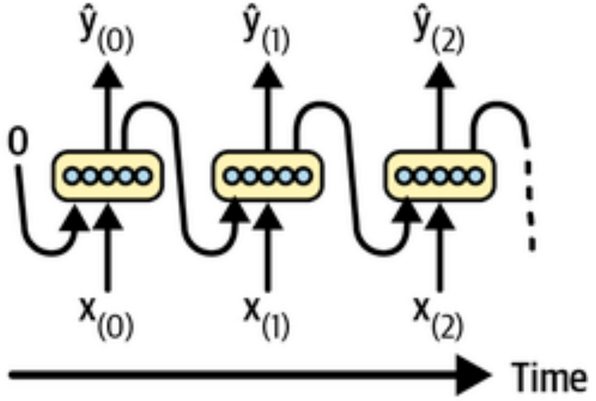
*Cada vector recibe:*

$X_{(t)}$ $\hat{y}_{(t-1)}$

### Recurrent Neurons and Layers




unrolling the network through



# RNN, LSTM y GRU

## Recurrent Neural Network

### Recurrent Neural Network

*Recibe entradas, produce una salida y envía esa salida a sí misma.*

*Cada vector recibe:*

$X_{(t)}$ $\hat{y}_{(t-1)}$

### Recurrent Neurons and Layers

unrolling the network through

$W_x$ *Matriz de entradas* $X_{(t)}$

$W_{\hat{y}}$ *Matriz de salidas en el paso del tiempo* $\hat{y}_{(t-1)}$

$$\hat{\mathbf{y}}_{(t)} = \phi\left(\mathbf{W}_x{}^{\intercal}\mathbf{x}_{(t)} + \mathbf{W}_{\hat{y}}{}^{\intercal}\hat{\mathbf{y}}_{(t-1)} + \mathbf{b}\right)$$

# RNN, LSTM y GRU

## Recurrent Neural Network

### Recurrent Neural Network

*Recibe entradas, produce una salida y envía esa salida a sí misma.*

*Cada vector recibe:*

$X_{(t)}$ $\hat{y}_{(t-1)}$

### Recurrent Neurons and Layers

unrolling the network through

$$\hat{\mathbf{Y}}_{(t)} = \phi\left(\mathbf{X}_{(t)}\mathbf{W}_x + \hat{\mathbf{Y}}_{(t-1)}\mathbf{W}_{\hat{y}} + \mathbf{b}\right)$$

$$= \phi\left(\begin{bmatrix}\mathbf{X}_{(t)} & \hat{\mathbf{Y}}_{(t-1)}\end{bmatrix}\mathbf{W} + \mathbf{b}\right) \text{ with } \mathbf{W} = \begin{bmatrix}\mathbf{W}_x \\ \mathbf{W}_{\hat{y}}\end{bmatrix}$$

# RNN, LSTM y GRU

## Recurrent Neural Network

**Recurrent Neural Network**

*Recibe entradas, produce una salida y envía esa salida a sí misma.*

*Cada vector recibe:*

$X_{(t)}$ $\hat{y}_{(t-1)}$

$$\hat{\mathbf{Y}}_{(t)} = \phi\left(\mathbf{X}_{(t)}\mathbf{W}_x + \hat{\mathbf{Y}}_{(t-1)}\mathbf{W}_{\hat{y}} + \mathbf{b}\right)$$

$$= \phi\left(\begin{bmatrix}\mathbf{X}_{(t)} & \hat{\mathbf{Y}}_{(t-1)}\end{bmatrix}\mathbf{W} + \mathbf{b}\right) \text{ with } \mathbf{W} = \begin{bmatrix}\mathbf{W}_x \\ \mathbf{W}_{\hat{y}}\end{bmatrix}$$

$\hat{y}_{(t)}$ *Es una matriz que contiene las salidas de la capa en el paso del tiempo.*

$X_{(t)}$ *Es una matriz que contiene las entradas para todos los registros (número de características)*

$W_x$ *Es una matriz que contiene los pesos de conexión para las entradas del paso de tiempo actual.*

$W_{\hat{y}}$ *Es una matriz que contiene los pesos de conexión para las salidas del paso de tiempo anterior.*

$b$ *Es un vector que contiene el terminó de sesgo de cada neurona.*

# RNN, LSTM y GRU

## Recurrent Neural Network

**Recurrent Neural Layer**

*Recibe entradas, produce una salida y envía esa salida a sí misma.*

*Cada vector recibe:*

$X_{(t)}$ $\hat{y}_{(t-1)}$

$$\hat{\mathbf{Y}}_{(t)} = \phi\left(\mathbf{X}_{(t)}\mathbf{W}_x + \hat{\mathbf{Y}}_{(t-1)}\mathbf{W}_{\hat{y}} + \mathbf{b}\right)$$

$$= \phi\left(\begin{bmatrix}\mathbf{X}_{(t)} & \hat{\mathbf{Y}}_{(t-1)}\end{bmatrix}\mathbf{W} + \mathbf{b}\right) \text{ with } \mathbf{W} = \begin{bmatrix}\mathbf{W}_x \\ \mathbf{W}_{\hat{y}}\end{bmatrix}$$

$W_x, W_{\hat{y}}$ *A menudo se concatenan verticalmente en una única matriz* $W$

$\left[X_{(t)}, \hat{y}_{(t-1)}\right]$ *Representan la concatenación horizontal de* $X_{(t)}, \hat{y}_{(t-1)}$

# RNN, LSTM y GRU

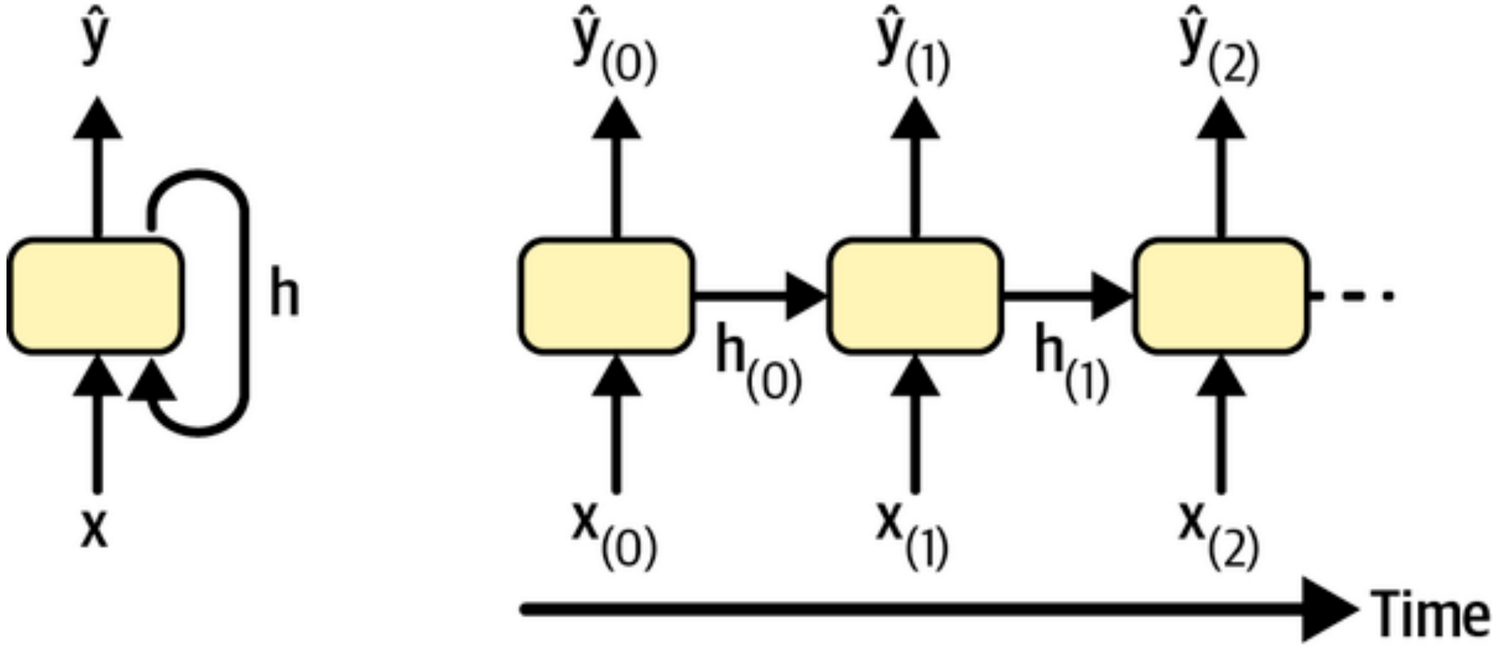
## Recurrent Neural Network

**Recurrent Neural Layer**

*Recibe entradas, produce una salida y envía esa salida a sí misma.*

*La salida en el paso del tiempo, es una función de todas las entradas anteriores. Por estos e dice que tiene memoria.*

**Memory cell** $h_{(t)} = f(x_{(t)}, h_{(t-1)})$ $h_{(t)} = tanh(x_{(t)}, h_{(t-1)})$



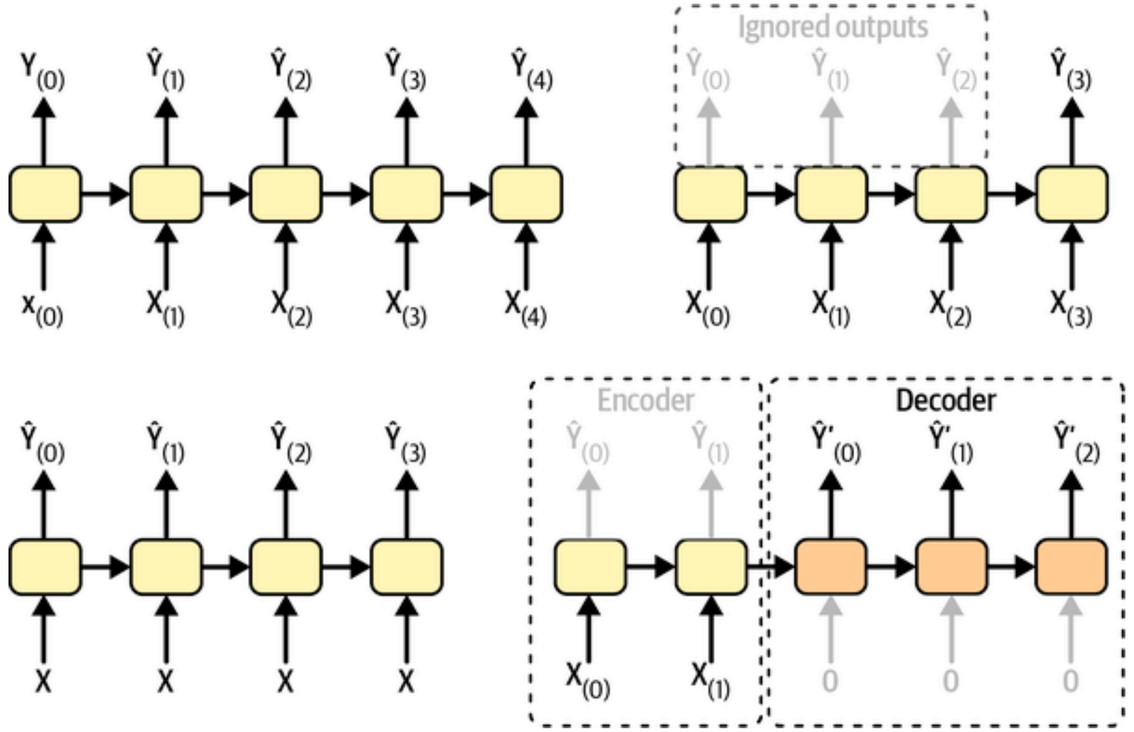
# RNN, LSTM y GRU

## Recurrent Neural Network

### Recurrent Neural Layer

*Recibe entradas, produce una salida y envía esa salida a sí misma.*

### Secuencia de entrada y salida



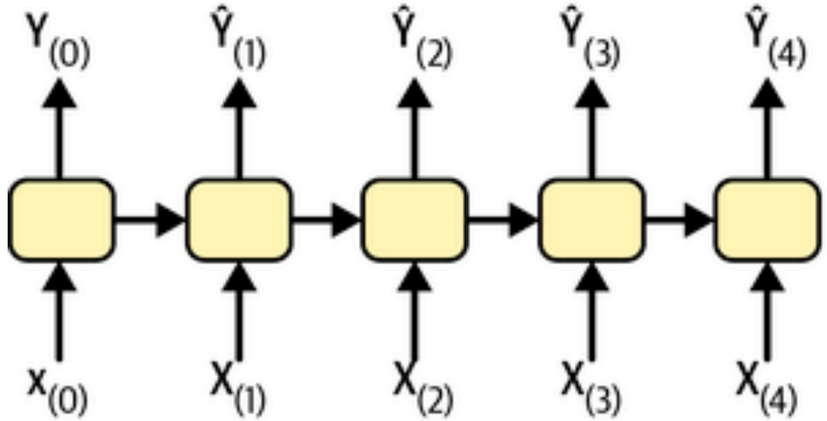
# RNN, LSTM y GRU

## Recurrent Neural Network

**Recurrent Neural Layer**

*Recibe entradas, produce una salida y envía esa salida a sí misma.*

**Secuencia de entrada y salida**




*Secuencia a secuencia*

↓

*Pronóstico de una serie temporal*

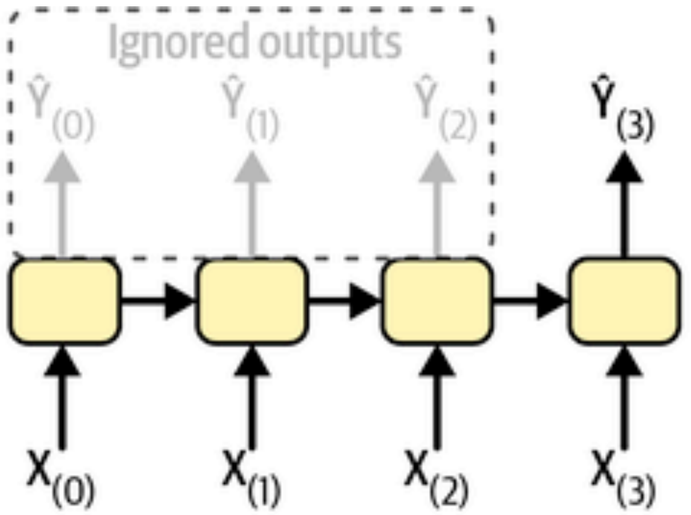
# RNN, LSTM y GRU

## Recurrent Neural Network

**Recurrent Neural Layer**

*Recibe entradas, produce una salida y envía esa salida a sí misma.*

**Secuencia de entrada y salida**




*Secuencia a vector*

*Secuencia de palabras, para pronosticar una reseña de película.*

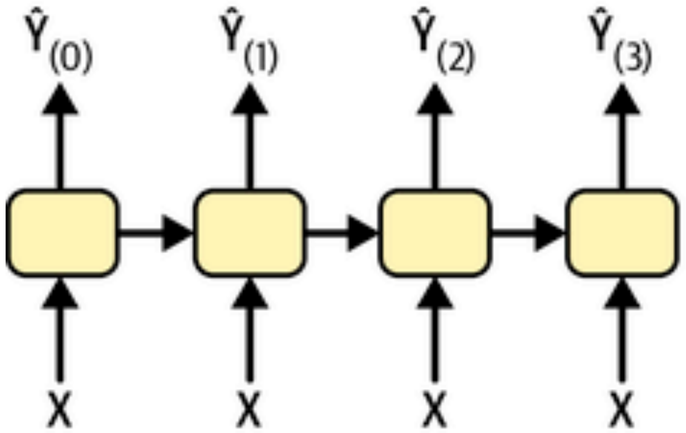
# RNN, LSTM y GRU

## Recurrent Neural Network

**Recurrent Neural Layer**

*Recibe entradas, produce una salida y envía esa salida a sí misma.*

**Secuencia de entrada y salida**




*Vector a secuencia*

*Entrada una imagen y salida el título de la imagen.*

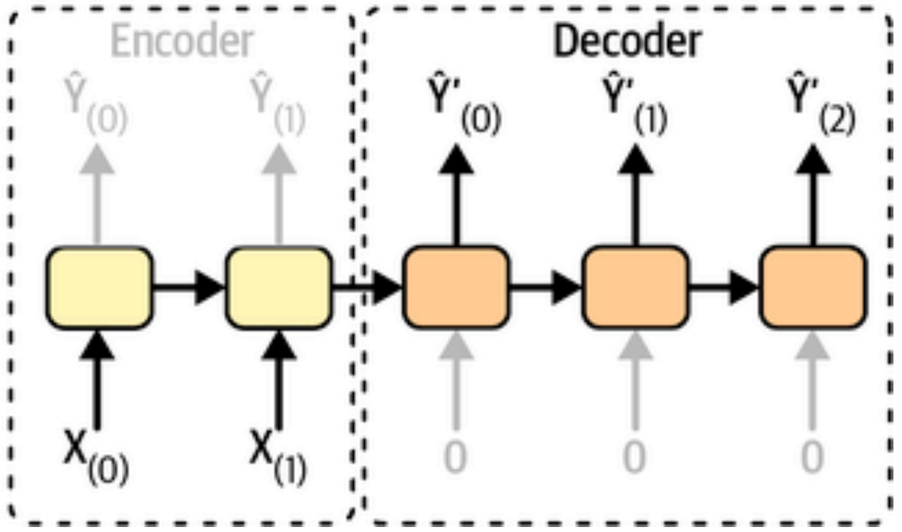
# RNN, LSTM y GRU

## Recurrent Neural Network

**Recurrent Neural Layer**

*Recibe entradas, produce una salida y envía esa salida a sí misma.*

**Secuencia de entrada y salida**




*Encoder-decoder*

*Encoder (secuencia a vector)*

*Decoder (Vector a secuencia)*

*Traducir una oración de un idioma a otro*

# RNN, LSTM y GRU

## Recurrent Neural Network

### Recurrent Neural Layer

*Recibe entradas, produce una salida y envía esa salida a sí misma.*

### Secuencia de entrada y salida

Recurrent Continuous Translation Models

**Recurrent Continuous Translation Models**

**Nal Kalchbrenner** **Phil Blunsom**
Department of Computer Science
University of Oxford
{nal.kalchbrenner,phil.blunsom}@cs.ox.ac.uk

**Abstract**

We introduce a class of probabilistic continuous translation models called Recurrent Continuous Translation Models that are purely based on continuous representations for words, phrases and sentences and do not rely on alignments or phrasal translation units. The models have a generation and a conditioning aspect. The generation of the translation is modelled with a target Recurrent Language Model, whereas the conditioning on the source sentence is modelled with a Convolutional Sentence Model. Through various experiments, we show first that our models obtain a perplexity with respect to gold translations that is > 43% lower than that of state-of-the-art alignment-based translation models. Secondly, we show that they are remarkably sensitive to the word order, syntax, and meaning of the source sentence despite lacking alignments. Finally we show that they match a state-of-the-art system when rescoring $n$-best lists of translations.

**1 Introduction**

In most statistical approaches to machine translation the basic units of translation are phrases that are composed of one or more words. A crucial component of translation systems are models that estimate translation probabilities for pairs of phrases, one phrase being from the source language and the other from the target language. Such models count phrase pairs and their occurrences as distinct if the surface forms of the phrases are distinct. Although distinct phrase pairs often share significant similarities, linguistic or otherwise, they do not share statistical weight in the models' estimation of their translation probabilities. Besides ignoring the similarity of phrase pairs, this leads to general sparsity issues. The estimation is sparse or skewed for the large number of rare or unseen phrase pairs, which grows exponentially in the length of the phrases, and the generalisation to other domains is often limited.

Continuous representations have shown promise at tackling these issues. Continuous representations for words are able to capture their morphological, syntactic and semantic similarity (Collobert and Weston, 2008). They have been applied in continuous language models demonstrating the ability to overcome sparsity issues and to achieve state-of-the-art performance (Bengio et al., 2003; Mikolov et al., 2010). Word representations have also shown a marked sensitivity to conditioning information (Mikolov and Zweig, 2012). Continuous representations for characters have been deployed in character-level language models demonstrating notable language generation capabilities (Sutskever et al., 2011). Continuous representations have also been constructed for phrases and sentences. The representations are able to carry similarity and task dependent information, e.g. sentiment, paraphrase or dialogue labels, significantly beyond the word level and to accurately predict labels for a highly diverse range of unseen phrases and sentences (Grefenstette et al., 2011; Socher et al., 2011; Socher et al., 2012; Hermann and Blunsom, 2013; Kalchbrenner and Blunsom, 2013).

Phrase-based continuous translation models were first proposed in (Schwenk et al., 2006) and re-



*Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1700–1709,
Seattle, Washington, USA, 18-21 October 2013. 

*Encoder-decoder*

↓

*Encoder (secuencia a vector)*

*Decoder (Vector a secuencia)*

*Traducir una oración de un idioma a otro*

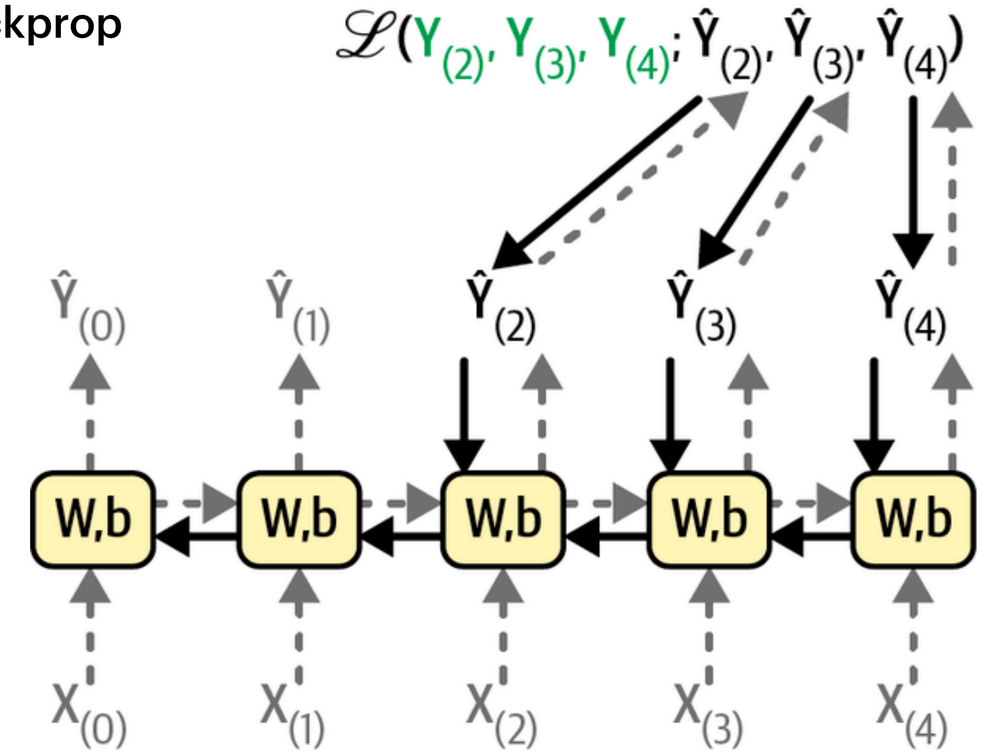
# RNN, LSTM y GRU

## Recurrent Neural Network

**Recurrent Neural Layer**

*Recibe entradas, produce una salida y envía esa salida a sí misma.*

**Backprop**



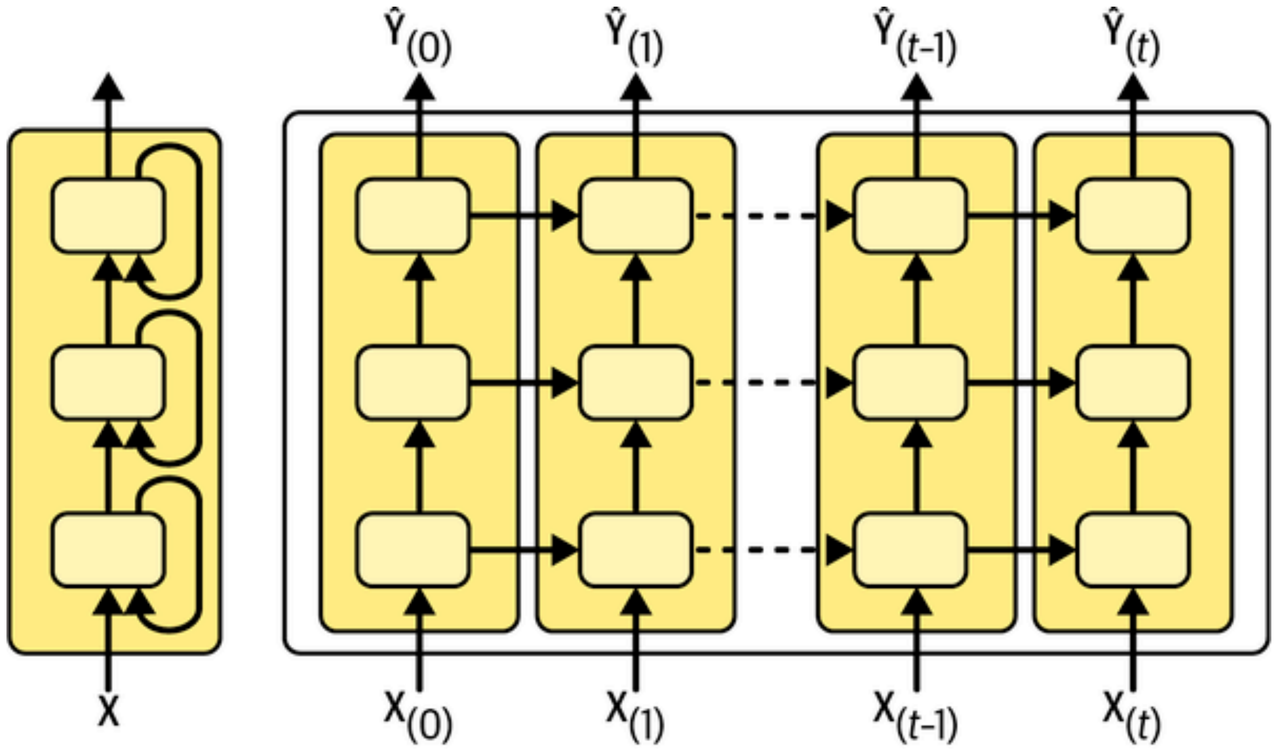
# RNN, LSTM y GRU

## Recurrent Neural Network

**Recurrent Neural Layer**

*Recibe entradas, produce una salida y envía esa salida a sí misma.*

**Deep RNN**

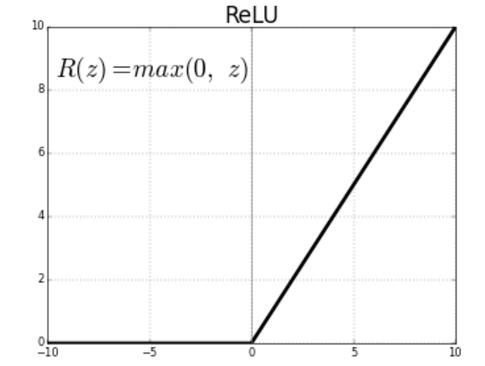


# RNN, LSTM y GRU

## Recurrent Neural Network

### Recurrent Neural Layer

*Recibe entradas, produce una salida y envía esa salida a sí misma.*




### Fighting the Unstable Gradients Problem

*Escenario donde explotan:* $W_2 > 1$

$$W_2 = 2$$

$$= Input_1 \cdot 2 \cdot 2 \cdot 2 \cdot 2$$

$$= Input_1 \cdot 2^4 = Input_1 \cdot 16$$

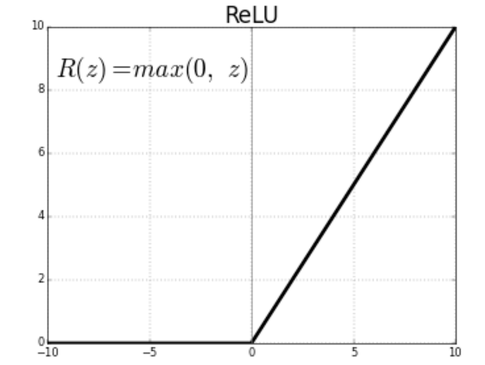
$$= Input_1 \cdot W_2^{UnrollTimes}$$

# RNN, LSTM y GRU

## Recurrent Neural Network

### Recurrent Neural Layer

*Recibe entradas, produce una salida y envía esa salida a sí misma.*




### Fighting the Unstable Gradients Problem

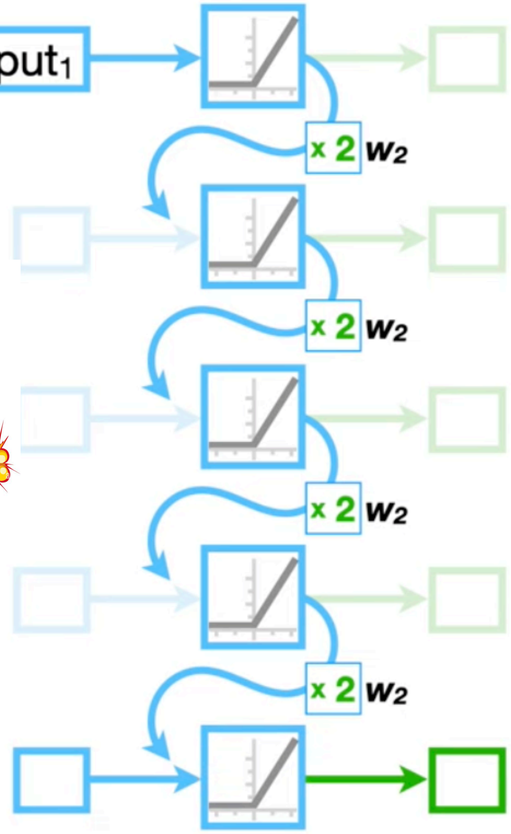
*Escenario donde explotan:* $W_2 > 1$

$$W_2 = 2$$

$$= Input_1 \cdot W_2^{UnrollTimes}$$

$$= Input_1 \cdot 2^{50} = 1.1258999e + 15$$


*Con 50 registros....*

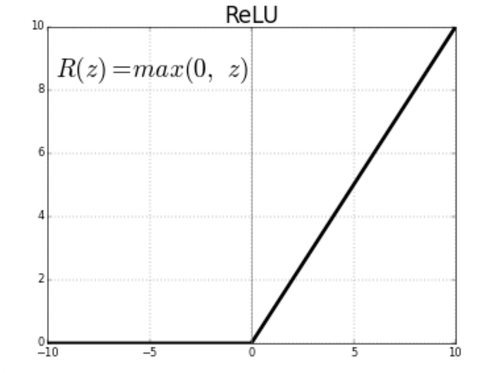


# RNN, LSTM y GRU

## Recurrent Neural Network

### Recurrent Neural Layer

*Recibe entradas, produce una salida y envía esa salida a sí misma.*




### Fighting the Unstable Gradients Problem

*Escenario donde desvanecen:* $W_2 < 1$

$$W_2 = 0.5$$

$$= Input_1 \cdot W_2^{UnrollTimes}$$

$$= Input_1 \cdot 0.5^{50} = Input_1 \cdot 0.00000000000000088817842$$

*Número muy cercano a cero...*

# RNN, LSTM y GRU

## Recurrent Neural Network

### Recurrent Neural Layer

*Recibe entradas, produce una salida y envía esa salida a sí misma.*

### Fighting the Unstable Gradients Problem

*La varianza de las salidas de cada capa es mucho mayor que la varianza de las entradas.*

saturación empeora por el hecho de que la función sigmoide

*Inicialización clásica:*

$$N(0,1)$$

*Función sigmoide:*

$$\mu = 0.5$$

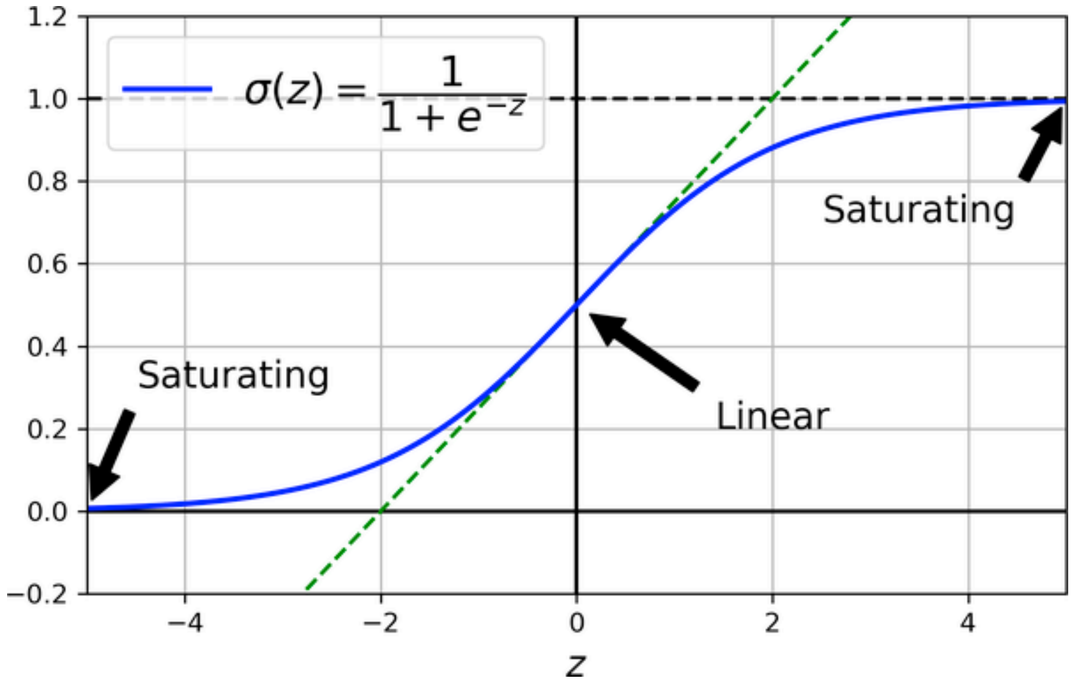
# RNN, LSTM y GRU

## Recurrent Neural Network

### Recurrent Neural Layer

*Recibe entradas, produce una salida y envía esa salida a sí misma.*

### Fighting the Unstable Gradients Problem




*La varianza de las salidas de cada capa es mucho mayor que la varianza de las entradas.*

*Inicialización de pesos con media 0 y desviación 1.*

# RNN, LSTM y GRU

## Recurrent Neural Network

### Recurrent Neural Layer

*Recibe entradas, produce una salida y envía esa salida a sí misma.*

### Fighting the Unstable Gradients Problem

*\Analogía:*

*si configuras la perilla de un amplificador de micrófono demasiado cerca de cero, la gente no escuchará tu voz, pero si la configuras demasiado cerca del máximo, tu voz se saturará y la gente no entenderá lo que estás diciendo.*

*Ahora imagine una cadena de amplificadores de este tipo: es necesario configurarlos correctamente para que su voz suene alta y clara al final de la cadena. Tu voz tiene que salir de cada amplificador con la misma amplitud con la que entró.*

# RNN, LSTM y GRU

## Recurrent Neural Network

### Recurrent Neural Layer

*Recibe entradas, produce una salida y envía esa salida a sí misma.*

### Fighting the Unstable Gradients Problem

**Understanding the difficulty of training deep feedforward neural networks**

**Xavier Glorot** **Yoshua Bengio**
DIRO, Université de Montréal, Montréal, Québec, Canada

**Abstract**

Whereas before 2006 it appears that deep multi-layer neural networks were not successfully trained, since then several algorithms have been shown to successfully train them, with experimental results showing the superiority of deeper vs less deep architectures. All these experimental results were obtained with new initialization or training mechanisms. Our objective here is to understand better why standard gradient descent from random initialization is doing so poorly with deep neural networks, to better understand these recent relative successes and help design better algorithms in the future. We first observe the influence of the non-linear activations functions. We find that the logistic sigmoid activation is unsuited for deep networks with random initialization because of its mean value, which can drive especially the top hidden layer into saturation. Surprisingly, we find that saturated units can move out of saturation by themselves, albeit slowly, and explaining the plateaus sometimes seen when training neural networks. We find that a new non-linearity that saturates less can often be beneficial. Finally, we study how activations and gradients vary across layers and during training, with the idea that training may be more difficult when the singular values of the Jacobian associated with each layer are far from 1. Based on these considerations, we propose a new initialization scheme that brings substantially faster convergence.

**1 Deep Neural Networks**

Deep learning methods aim at learning feature hierarchies with features from higher levels of the hierarchy formed by the composition of lower level features. They include learning methods for a wide array of *deep architectures*, including neural networks with many hidden layers (Vincent et al., 2008) and graphical models with many levels of hidden variables (Hinton et al., 2006), among others (Zhu et al., 2009; Weston et al., 2008). Much attention has recently been devoted to them (see (Bengio, 2009) for a review), because of their theoretical appeal, inspiration from biology and human cognition, and because of empirical success in vision (Ranzato et al., 2007; Larochelle et al., 2007; Vincent et al., 2008) and natural language processing (NLP) (Collobert & Weston, 2008; Mnih & Hinton, 2009). Theoretical results reviewed and discussed by Bengio (2009), suggest that in order to learn the kind of complicated functions that can represent high-level abstractions (e.g. in vision, language, and other AI-level tasks), one may need *deep architectures*.

Most of the recent experimental results with deep architecture are obtained with models that can be turned into deep supervised neural networks, but with initialization or training schemes different from the classical feedforward neural networks (Rumelhart et al., 1986). Why are these new algorithms working so much better than the standard random initialization and gradient-based optimization of a supervised training criterion? Part of the answer may be found in recent analyses of the effect of unsupervised pre-training (Erhan et al., 2009), showing that it acts as a regularizer that initializes the parameters in a "better" basin of attraction of the optimization procedure, corresponding to an apparent local minimum associated with better generalization. But earlier work (Bengio et al., 2007) had shown that even a purely supervised but greedy layer-wise procedure would give better results. So here instead of focusing on what unsupervised pre-training or semi-supervised criteria bring to deep architectures, we focus on analyzing what may be going wrong with good old (but deep) multi-layer neural networks.

Our analysis is driven by investigative experiments to monitor activations (watching for saturation of hidden units) and gradients, across layers and across training iterations. We also evaluate the effects on these of choices of activation function (with the idea that it might affect saturation) and initialization procedure (since unsupervised pre-training is a particular form of initialization and it has a drastic impact).

Appearing in Proceedings of the $13^{th}$ International Conference on Artificial Intelligence and Statistics (AISTATS) 2010, Chia Laguna Resort, Sardinia, Italy. Volume 9 of JMLR: W&CP 9. 



*Xavier Glorot*
*Yoshua Bengio*

**3 Effect of Activation Functions and Saturation During Training**

Two things we want to avoid and that can be revealed from the evolution of activations is excessive saturation of activation functions on one hand (then gradients will not propagate well), and overly linear units (they will not compute something interesting).

# RNN, LSTM y GRU

## Recurrent Neural Network

### Recurrent Neural Layer

*Recibe entradas, produce una salida y envía esa salida a sí misma.*

### Fighting the Unstable Gradients Problem

Dropout improves Recurrent Neural Networks for Handwriting Recognition

Vu Pham*†, Théodore Bluche*‡, Christopher Kermorvant*, and Jérôme Louradour*
* A2iA, 39 rue de la Bienfaisance, 75008 - Paris - France
† SUTD, 20 Dover Drive, Singapore
‡LIMSI CNRS, Spoken Language Processing Group, Orsay, France



***Abstract*—Recurrent neural networks (RNNs) with Long Short-Term memory cells currently hold the best known results in unconstrained handwriting recognition. We show that their performance can be greatly improved using *dropout* - a recently proposed regularization method for deep architectures. While previous works showed that dropout gave superior performance in the context of convolutional networks, it had never been applied to RNNs. In our approach, dropout is carefully used in the network so that it does not affect the recurrent connections, hence the power of RNNs in modeling sequences is preserved. Extensive experiments on a broad range of handwritten databases confirm the effectiveness of dropout on deep architectures even when the network mainly consists of recurrent and shared connections.**

***Keywords*—Recurrent Neural Networks, Dropout, Handwriting Recognition**

I. Introduction

Unconstrained offline handwriting recognition is the problem of recognizing long sequences of text when only an image of the text is available. The only constraint in such a setting is that the text is written in a given language. Usually a pre-processing module is used to extract image snippets, each contains one single word or line, which are then fed into the recognizer. A handwriting recognizer, therefore, is in charge of recognizing one single line of text at a time. Generally, such a recognizer should be able to detect the correlation between characters in the sequence, so it has more information about the local context and presumably provides better performance. Readers are referred to [1] for an extensive review of handwriting recognition systems.

Early works typically use a Hidden Markov Model (HMM) [2] or an HMM-neural network hybrid system [3], [4] for the recognizer. However, the hidden states of HMMs follow a first-order Markov chain, hence they cannot handle long-term dependencies in sequences. Moreover, at each time step, HMMs can only select one hidden state, hence an HMM with $n$ hidden states can typically carry only $\log(n)$ bits of information about its dynamics [5].

Recurrent neural networks (RNNs) do not have such limitations and were shown to be very effective in sequence modeling. With their recurrent connections, RNNs can, in principle, store representations of past input events in form of activations, allowing them to model long sequences with complex structures. RNNs are inherently deep in time and can have many layers, both make training parameters a difficult optimization problem. The burden of exploding and vanishing gradient was the reason for the lack of practical applications of RNNs until recently [6], [7].

Lately, an advance in designing RNNs was proposed, namely Long Short-Term Memory (LSTM) cells. LSTM are carefully designed recurrent neurons which gave superior performance in a wide range of sequence modeling problems. In fact, RNNs enhanced by LSTM cells [8] won several important contests [9], [10], [11] and currently hold the best known results in handwriting recognition.

Meanwhile, in the emerging deep learning movement, dropout was used to effectively prevent deep neural networks with lots of parameters from overfitting. It is shown to be effective with deep convolutional networks [12], [13], [14], feed-forward networks [15], [16], [17] but, to the best of our knowledge, has never been applied to RNNs. Moreover, dropout was typically applied only at fully-connected layers [12], [18], even in convolutional networks [13]. In this work, we show that dropout can also be used in RNNs at some certain layers which are not necessarily fully-connected. The choice of applying dropout is carefully made so that it does not affect the recurrent connections, therefore without reducing the ability of RNNs to model long sequences.

Due to the impressive performance of dropout, some extensions of this technique were proposed, including DropConnect [18], Maxout networks [19], and an approximate approach for fast training with dropout [20]. In [18], a theoretical generalization bound of dropout was also derived. In this work, we only consider the original idea of dropout [12].

Section II presents the RNN architecture designed for handwriting recognition. Dropout is then adapted for this architecture as described in Section III. Experimental results are given and analyzed in Section IV, while the last section is dedicated for conclusions.

II. Recurrent neural networks for Handwriting recognition

The recognition system considered in this work is depicted in Fig. 1. The input image is divided into blocks of size $2 \times 2$ and fed into four LSTM layers which scan the input in different directions indicated by corresponding arrows. The output of each LSTM layer is separately fed into convolutional layers of 6 features with filter size $2 \times 4$. This convolutional layer is applied without overlaping nor biases. It can be seen as a subsampling step, with trainable weights rather than a deterministic subsampling function. The activations of 4 convolutional layers are then summed element-wise and squashed by the hyperbolic tangent (tanh) function. This process is repeated twice with different filter sizes and numbers

*Vu Pham∗†, Theodore Bluche´ ∗‡, Christopher Kermorvant∗ , and Jer´ ome Louradour*

Note that previous works about dropout seem to favor rectified linear units (ReLU) [13] over *tanh* or *sigmoid* for the network nonlinearity since it provides better covergence rate. In our experiments, however, we find out that ReLU can not give good performance in LSTM cells, hence we keep *tanh* for the LSTM cells and *sigmoid* for the gates.

Pontificia Universidad JAVERIANA Colombia

# RNN, LSTM y GRU

## Recurrent Neural Network

### Recurrent Neural Layer

*Recibe entradas, produce una salida y envía esa salida a sí misma.*

### Fighting the Unstable Gradients Problem

arXiv:1504.00941v2 [cs.NE] 7 Apr 2015

**A Simple Way to Initialize Recurrent Networks of Rectified Linear Units**

Quoc V. Le, Navdeep Jaitly, Geoffrey E. Hinton
Google

**Abstract**

Learning long term dependencies in recurrent networks is difficult due to vanishing and exploding gradients. To overcome this difficulty, researchers have developed sophisticated optimization techniques and network architectures. In this paper, we propose a simpler solution that use recurrent neural networks composed of rectified linear units. Key to our solution is the use of the identity matrix or its scaled version to initialize the recurrent weight matrix. We find that our solution is comparable to a standard implementation of LSTMs on our four benchmarks: two toy problems involving long-range temporal structures, a large language modeling problem and a benchmark speech recognition problem.

**1 Introduction**

Recurrent neural networks (RNNs) are very powerful dynamical systems and they are the natural way of using neural networks to map an input sequence to an output sequence, as in speech recognition and machine translation, or to predict the next term in a sequence, as in language modeling. However, training RNNs by using back-propagation through time [30] to compute error-derivatives can be difficult. Early attempts suffered from vanishing and exploding gradients [15] and this meant that they had great difficulty learning long-term dependencies. Many different methods have been proposed for overcoming this difficulty.

A method that has produced some impressive results [23, 24] is to abandon stochastic gradient descent in favor of a much more sophisticated Hessian-Free (HF) optimization method. HF operates on large mini-batches and is able to detect promising directions in the weight-space that have very small gradients but even smaller curvature. Subsequent work, however, suggested that similar results could be achieved by using stochastic gradient descent with momentum provided the weights were initialized carefully [34] and large gradients were clipped [28]. Further developments of the HF approach look promising [35, 25] but are much harder to implement than popular simple methods such as stochastic gradient descent with momentum [34] or adaptive learning rates for each weight that depend on the history of its gradients [5, 14].

The most successful technique to date is the Long Short Term Memory (LSTM) Recurrent Neural Network which uses stochastic gradient descent, but changes the hidden units in such a way that the backpropagated gradients are much better behaved [16]. LSTM replaces logistic or tanh hidden units with "memory cells" that can store an analog value. Each memory cell has its own input and output gates that control when inputs are allowed to add to the stored analog value and when this value is allowed to influence the output. These gates are logistic units with their own learned weights on connections coming from the input and also the memory cells at the previous time-step. There is also a forget gate with learned weights that controls the rate at which the analog value stored in the memory cell decays. For periods when the input and output gates are off and the forget gate is not causing decay, a memory cell simply holds its value over time so the gradient of the error w.r.t. its stored value stays constant when backpropagated over those periods.

1

*Quoc V. Le, Navdeep Jaitly, Geoffrey E. Hinton*

Recent research on deep feedforward networks has also produced some impressive results [19, 3] and there is now a consensus that for deep networks, rectified linear units (ReLUs) are easier to train than the logistic or tanh units that were used for many years [27, 40]. At first sight, ReLUs seem inappropriate for RNNs because they can have very large outputs so they might be expected to be far more likely to explode than units that have bounded values. A second aim of this paper is to explore whether ReLUs can be made to work well in RNNs and whether the ease of optimizing them in feedforward nets transfers to RNNs.

# RNN, LSTM y GRU

## Long Short Term Memory

### LSTM cells

*Recibe entradas, produce una salida y envía esa salida a sí misma, con dos vectores de estado y puertas*

**Pros**

1. *Converge más rápido*
2. *Detecta patrones de largo plazo*
3. *El estado se divide en dos vectores* $h_{(t)}$ $C_{(t)}$

*Keras tiene una optimización*

*Short-term state*

$h_{(t)}$

*Long-term state*

$C_{(t)}$

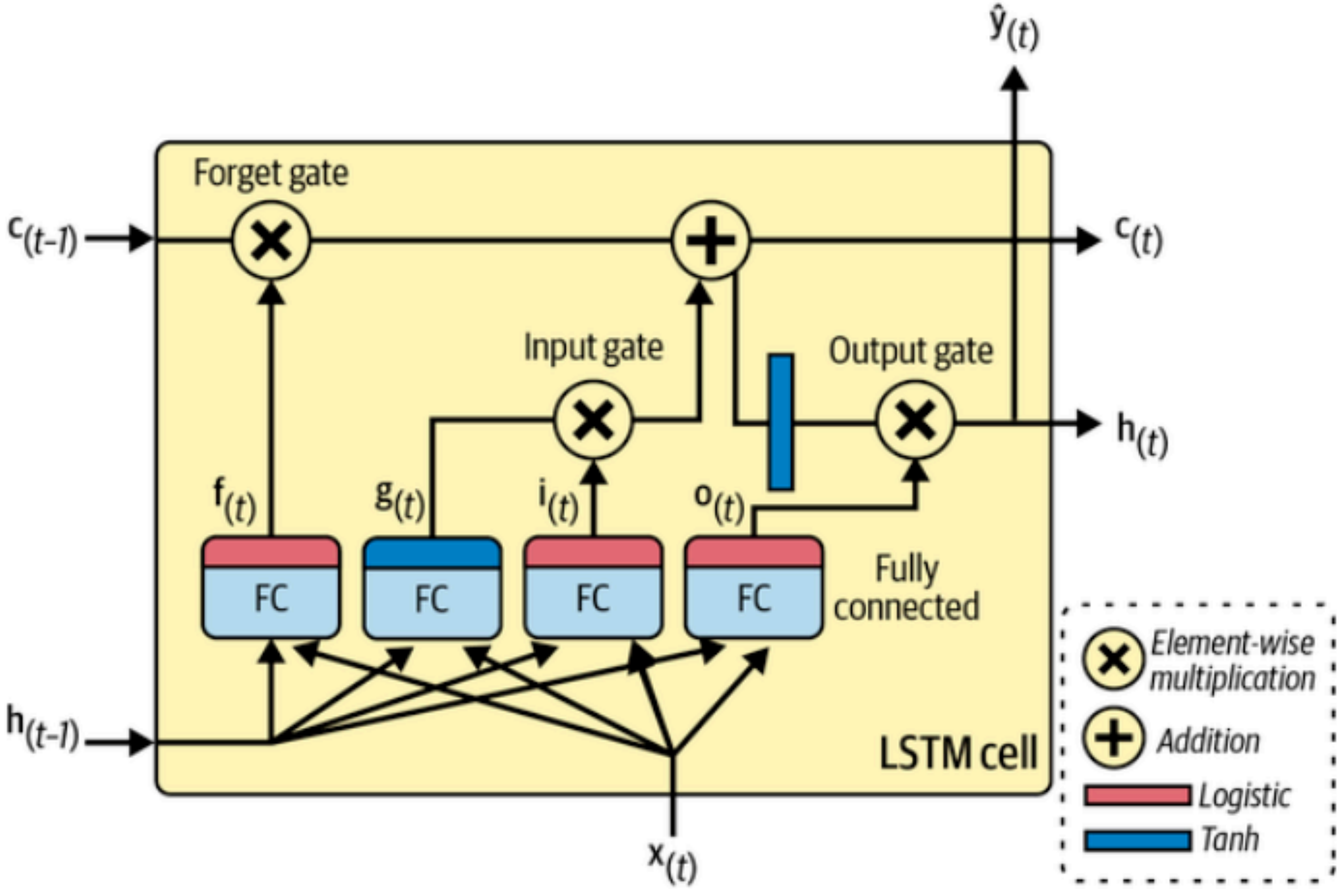
proposed in 1997[11] by Sepp Hochreiter

# RNN, LSTM y GRU

## Long Short Term Memory

### LSTM cells

*Recibe entradas, produce una salida y envía esa salida a sí misma, con dos vectores de estado y puertas*



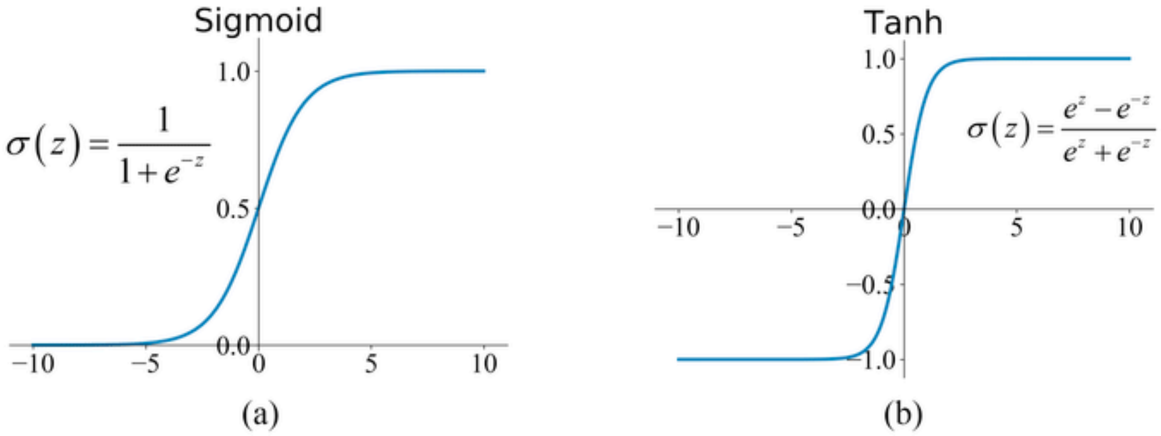
# RNN, LSTM y GRU

## Long Short Term Memory

### LSTM cells

*Recibe entradas, produce una salida y envía esa salida a sí misma, con dos vectores de estado y puertas*

**Funciones dentro de la celda LSTM**



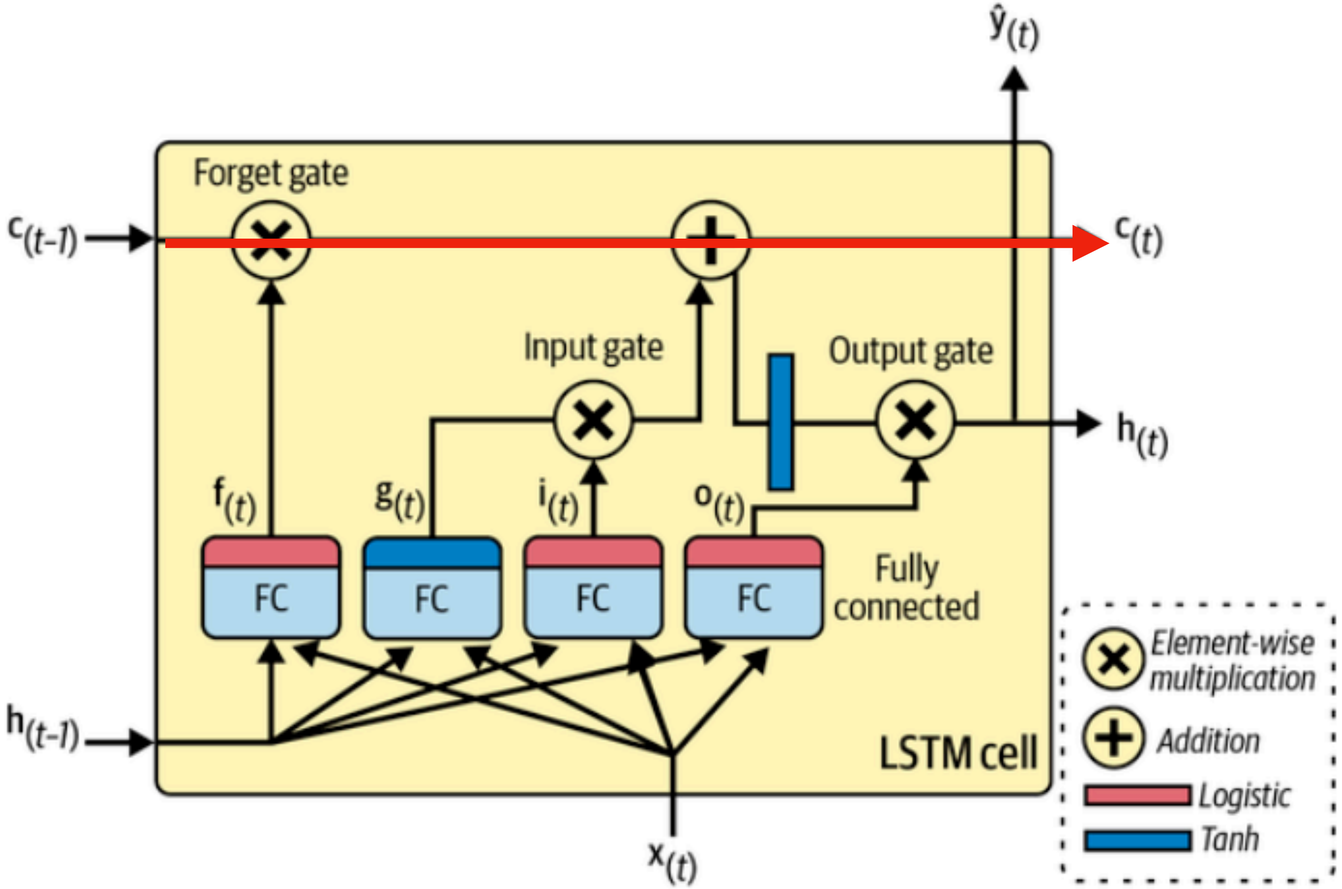
# RNN, LSTM y GRU

## Long Short Term Memory

### LSTM cells

*Recibe entradas, produce una salida y envía esa salida a sí misma, con dos vectores de estado y puertas*

*Memoria de largo plazo*



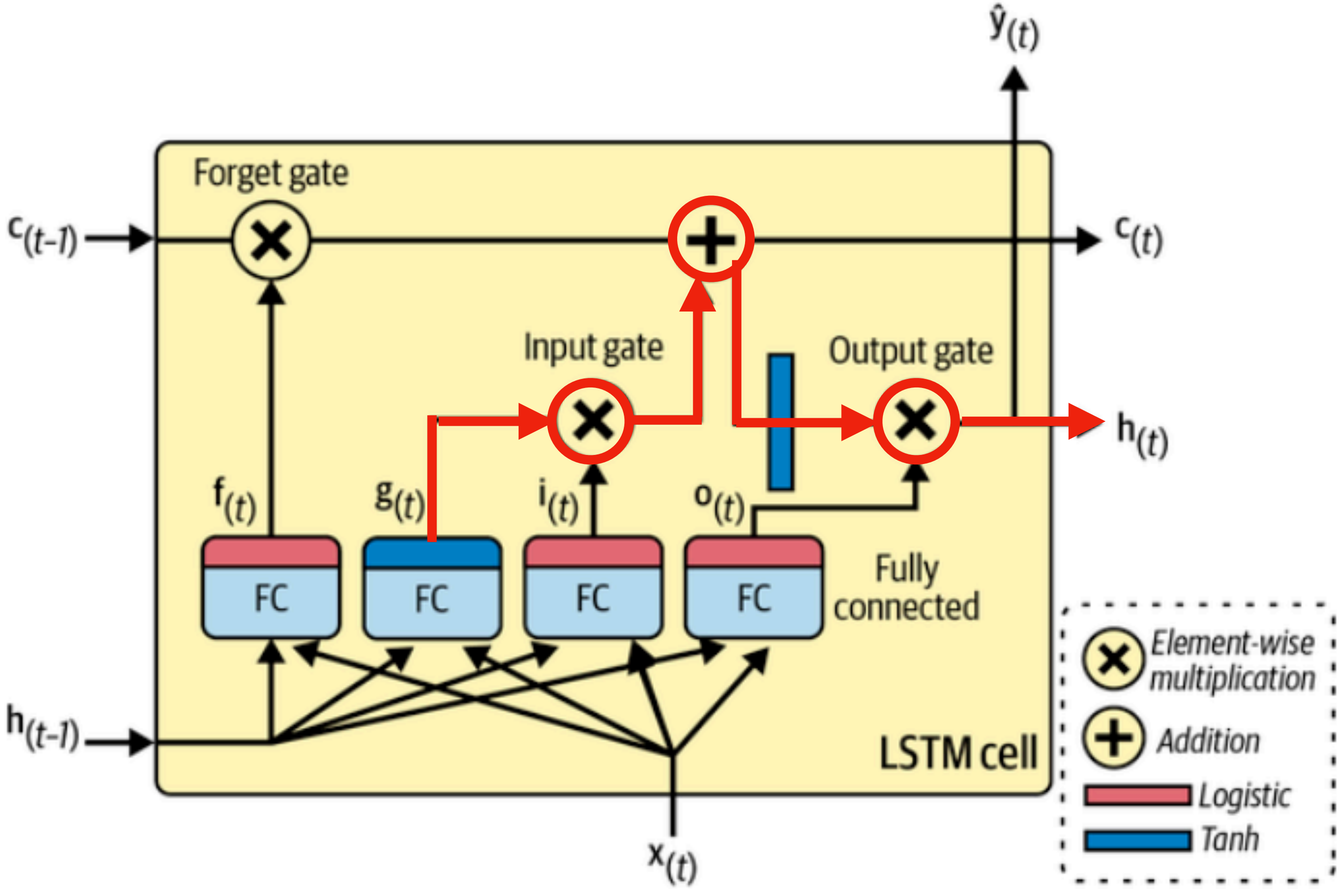
# RNN, LSTM y GRU

## Long Short Term Memory

### LSTM cells

*Recibe entradas, produce una salida y envía esa salida a sí misma, con dos vectores de estado y puertas*

*Memoria de corto plazo*



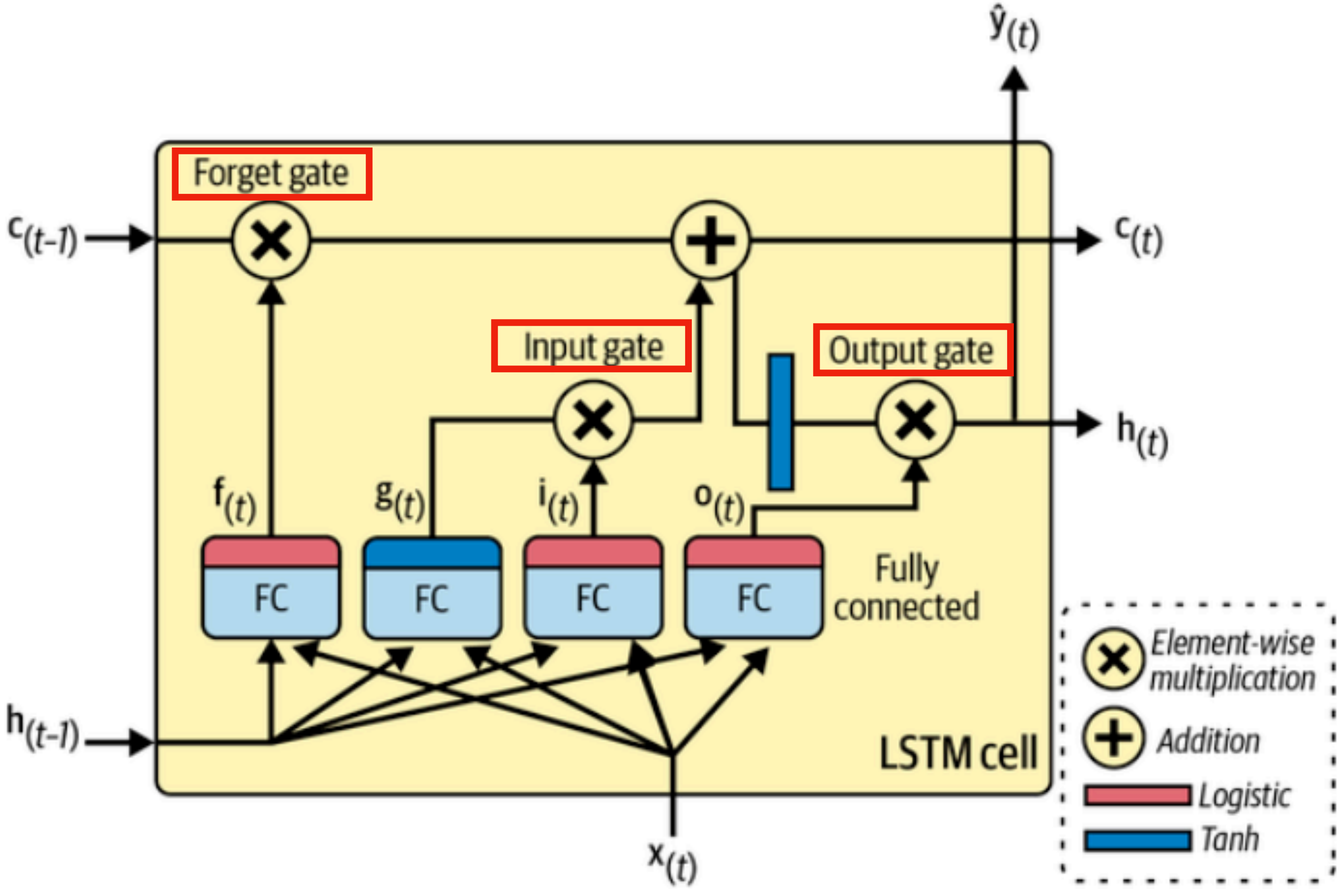
# RNN, LSTM y GRU

## Long Short Term Memory

### LSTM cells

*Recibe entradas, produce una salida y envía esa salida a sí misma, con dos vectores de estado y puertas*

*Puertas/gates*



# RNN, LSTM y GRU

## Long Short Term Memory

### LSTM cells

*Recibe entradas, produce una salida y envía esa salida a sí misma, con dos vectores de estado y puertas*

**Puertas**

| | |
|---|---|
| *Puerta de olvido* → | *Determina que porcentaje de de la memoria de largo plazo se recuerda* |
| *Puerta de entrada* → | *Determina cómo debemos actualizar la memoria de largo plazo.* |
| *Puerta de salida* → | *Determina la nueva memoria de corto plazo, dejándolo como output* |

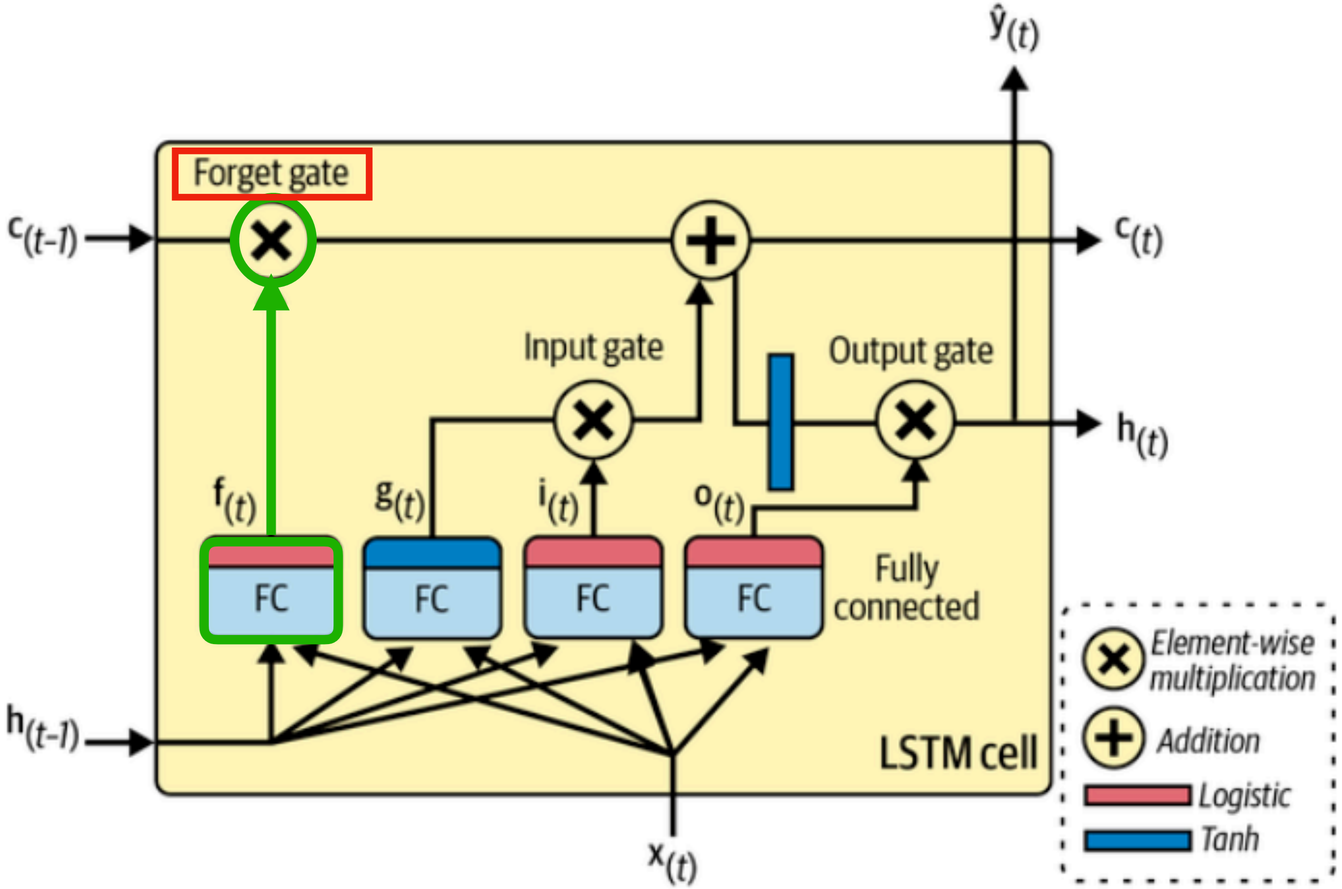
# RNN, LSTM y GRU

## Long Short Term Memory

### LSTM cells

*Recibe entradas, produce una salida y envía esa salida a sí misma, con dos vectores de estado y puertas*

**Porcentaje de memoria de largo plazo a recordar**



# RNN, LSTM y GRU

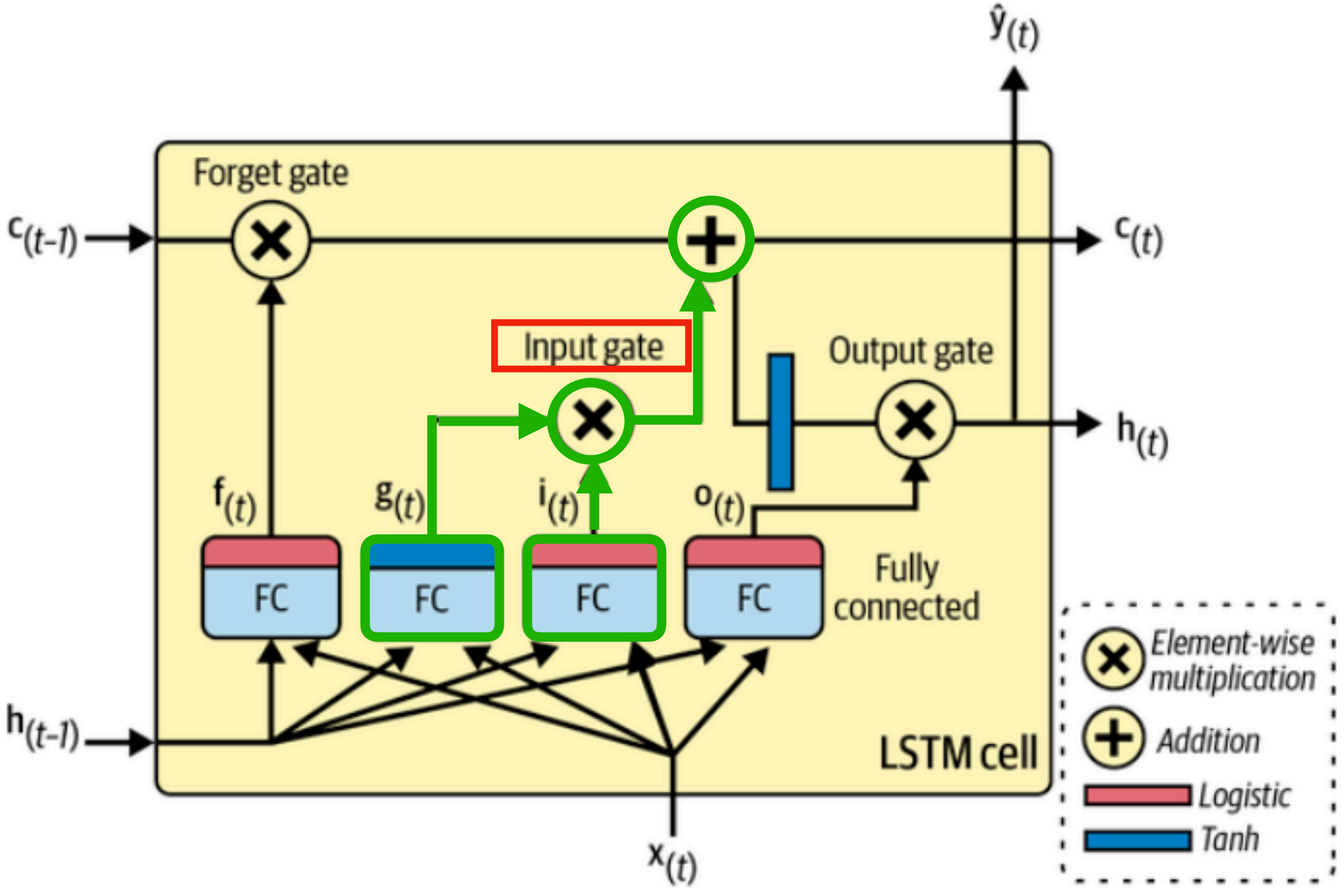
## Long Short Term Memory

### LSTM cells

*Recibe entradas, produce una salida y envía esa salida a sí misma, con dos vectores de estado y puertas*

**Porcentaje memoria potencial a recordar**


**Memoria potencial a recordar (largo plazo)**

# RNN, LSTM y GRU

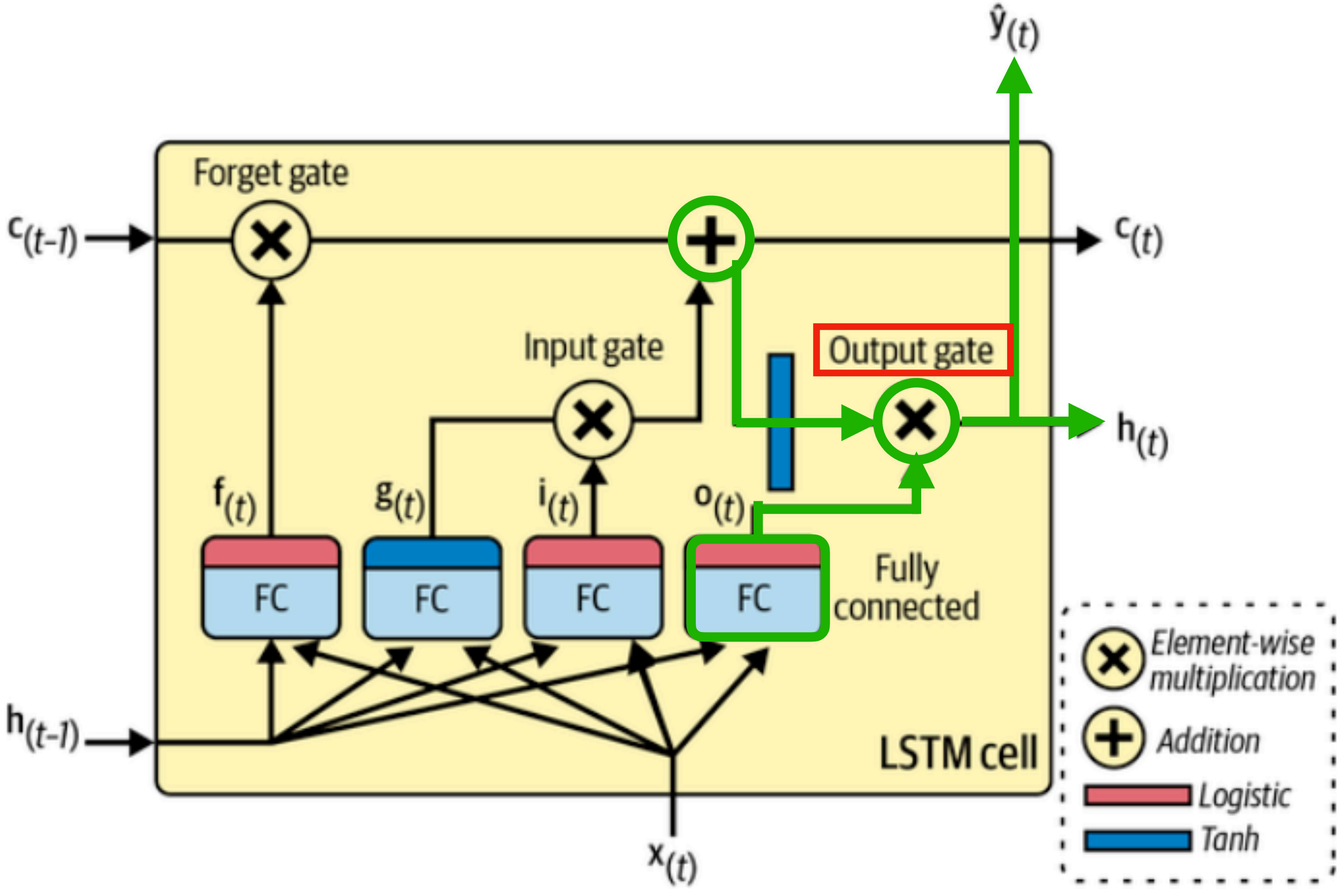
## Long Short Term Memory

### LSTM cells

*Recibe entradas, produce una salida y envía esa salida a sí misma, con dos vectores de estado y puertas*

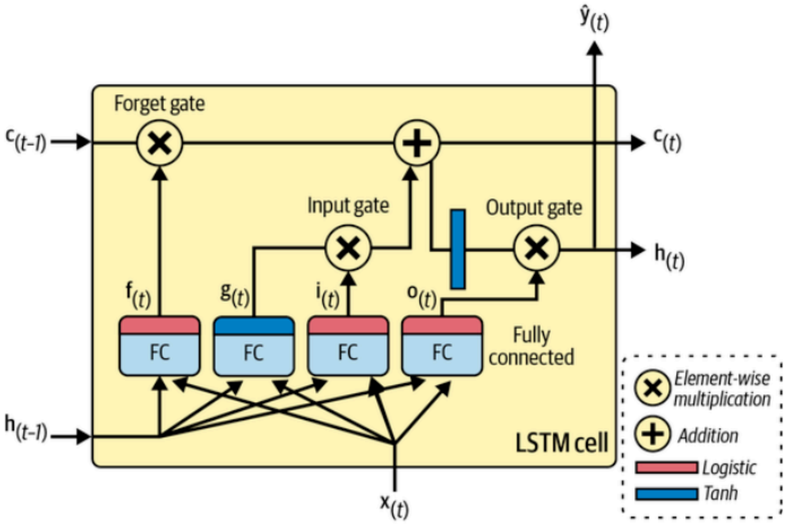
**Porcentaje de memoria a recordar**


**Memoria potencial a recordar (corto plazo)**



# RNN, LSTM y GRU

## Long Short Term Memory




## LSTM cells

*Recibe entradas, produce una salida y envía esa salida a sí misma, con dos vectores de estado y puertas*

**Funciones dentro de la celda LSTM**

$sigmoide = f_{(t)}$ *% de memoria de largo plazo a recordar*

$tanh = g_{(t)}$ *Memoria potencial de largo plazo*

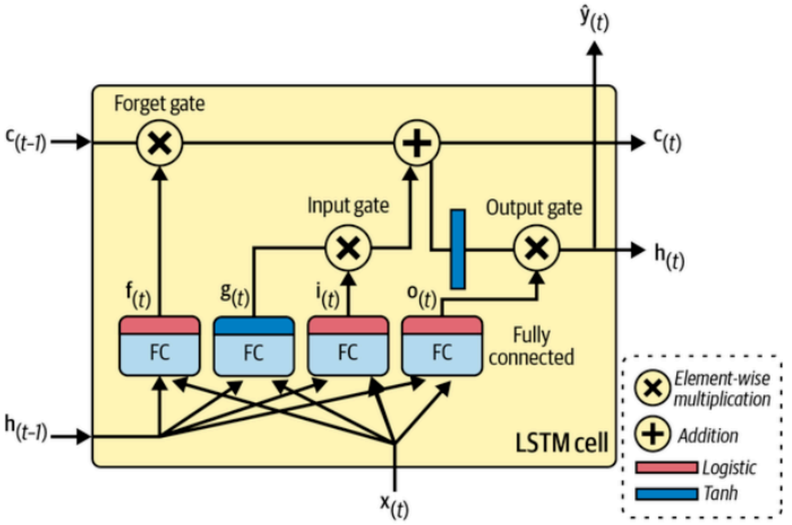
$sigmoide = i_{(t)}$ *% Memoria potencial de largo plazo*

$sigmoide = o_{(t)}$ *% Memoria potencial de corto plazo*

$tanh$ *Memoria potencial de corto plazo*

# RNN, LSTM y GRU

## Long Short Term Memory




## LSTM cells

*Recibe entradas, produce una salida y envía esa salida a sí misma, con dos vectores de estado y puertas*

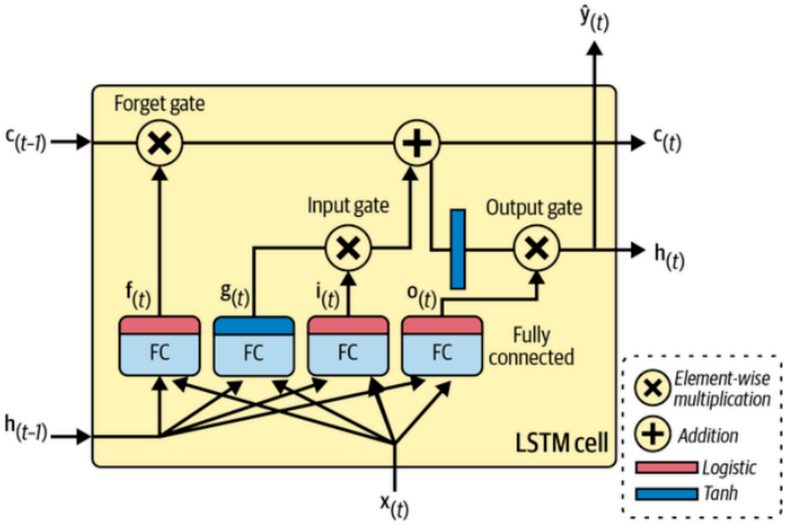
**Funciones dentro de la celda LSTM**

**Computo**

$$
\begin{aligned}
\mathbf{i}_{(t)} &= \sigma\left(\mathbf{W}_{xi}{}^{\intercal}\mathbf{x}_{(t)} + \mathbf{W}_{hi}{}^{\intercal}\mathbf{h}_{(t-1)} + \mathbf{b}_i\right) \\
\mathbf{f}_{(t)} &= \sigma\left(\mathbf{W}_{xf}{}^{\intercal}\mathbf{x}_{(t)} + \mathbf{W}_{hf}{}^{\intercal}\mathbf{h}_{(t-1)} + \mathbf{b}_f\right) \\
\mathbf{o}_{(t)} &= \sigma\left(\mathbf{W}_{xo}{}^{\intercal}\mathbf{x}_{(t)} + \mathbf{W}_{ho}{}^{\intercal}\mathbf{h}_{(t-1)} + \mathbf{b}_o\right) \\
\mathbf{g}_{(t)} &= \tanh\left(\mathbf{W}_{xg}{}^{\intercal}\mathbf{x}_{(t)} + \mathbf{W}_{hg}{}^{\intercal}\mathbf{h}_{(t-1)} + \mathbf{b}_g\right) \\
\mathbf{c}_{(t)} &= \mathbf{f}_{(t)} \otimes \mathbf{c}_{(t-1)} + \mathbf{i}_{(t)} \otimes \mathbf{g}_{(t)} \\
\mathbf{y}_{(t)} &= \mathbf{h}_{(t)} = \mathbf{o}_{(t)} \otimes \tanh\left(\mathbf{c}_{(t)}\right)
\end{aligned}
$$

# RNN, LSTM y GRU

## Long Short Term Memory




## LSTM cells

*Recibe entradas, produce una salida y envía esa salida a sí misma, con dos vectores de estado y puertas*

**Funciones dentro de la celda LSTM**

**Computo**

$$
\begin{aligned}
\mathbf{i}_{(t)} &= \sigma\left(\mathbf{W}_{xi}{}^{\intercal}\mathbf{x}_{(t)} + \mathbf{W}_{hi}{}^{\intercal}\mathbf{h}_{(t-1)} + \mathbf{b}_i\right) \\
\mathbf{f}_{(t)} &= \sigma\left(\mathbf{W}_{xf}{}^{\intercal}\mathbf{x}_{(t)} + \mathbf{W}_{hf}{}^{\intercal}\mathbf{h}_{(t-1)} + \mathbf{b}_f\right) \\
\mathbf{o}_{(t)} &= \sigma\left(\mathbf{W}_{xo}{}^{\intercal}\mathbf{x}_{(t)} + \mathbf{W}_{ho}{}^{\intercal}\mathbf{h}_{(t-1)} + \mathbf{b}_o\right) \\
\mathbf{g}_{(t)} &= \tanh\left(\mathbf{W}_{xg}{}^{\intercal}\mathbf{x}_{(t)} + \mathbf{W}_{hg}{}^{\intercal}\mathbf{h}_{(t-1)} + \mathbf{b}_g\right) \\
\mathbf{c}_{(t)} &= \mathbf{f}_{(t)} \otimes \mathbf{c}_{(t-1)} + \mathbf{i}_{(t)} \otimes \mathbf{g}_{(t)} \\
\mathbf{y}_{(t)} &= \mathbf{h}_{(t)} = \mathbf{o}_{(t)} \otimes \tanh\left(\mathbf{c}_{(t)}\right)
\end{aligned}
$$

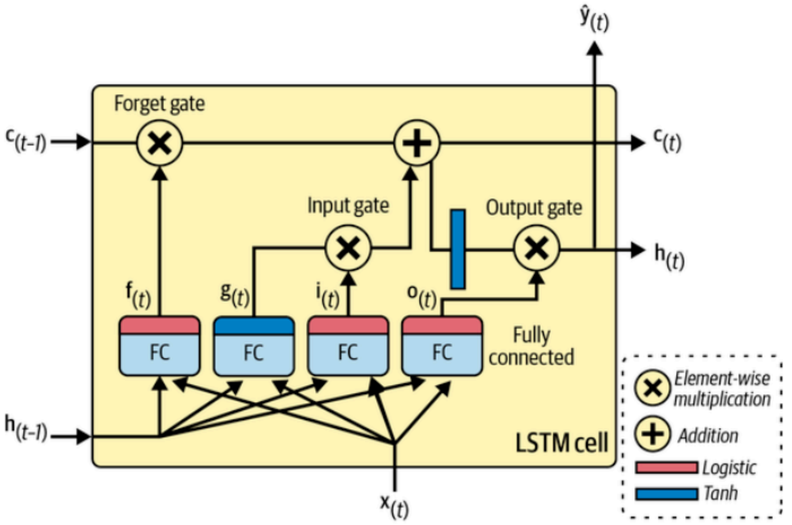
$W_{xi}, W_{xf}, W_{xo}, W_{xg}$

*Matriz de pesos de cada capa*

*4 capas conectadas al input* $X_{(t)}$

# RNN, LSTM y GRU

## Long Short Term Memory




## LSTM cells

*Recibe entradas, produce una salida y envía esa salida a sí misma, con dos vectores de estado y puertas*

**Funciones dentro de la celda LSTM**

**Computo**

$$
\begin{aligned}
\mathbf{i}_{(t)} &= \sigma\left(\mathbf{W}_{xi}{}^{\intercal}\mathbf{x}_{(t)} + \mathbf{W}_{hi}{}^{\intercal}\mathbf{h}_{(t-1)} + \mathbf{b}_i\right) \\
\mathbf{f}_{(t)} &= \sigma\left(\mathbf{W}_{xf}{}^{\intercal}\mathbf{x}_{(t)} + \mathbf{W}_{hf}{}^{\intercal}\mathbf{h}_{(t-1)} + \mathbf{b}_f\right) \\
\mathbf{o}_{(t)} &= \sigma\left(\mathbf{W}_{xo}{}^{\intercal}\mathbf{x}_{(t)} + \mathbf{W}_{ho}{}^{\intercal}\mathbf{h}_{(t-1)} + \mathbf{b}_o\right) \\
\mathbf{g}_{(t)} &= \tanh\left(\mathbf{W}_{xg}{}^{\intercal}\mathbf{x}_{(t)} + \mathbf{W}_{hg}{}^{\intercal}\mathbf{h}_{(t-1)} + \mathbf{b}_g\right) \\
\mathbf{c}_{(t)} &= \mathbf{f}_{(t)} \otimes \mathbf{c}_{(t-1)} + \mathbf{i}_{(t)} \otimes \mathbf{g}_{(t)} \\
\mathbf{y}_{(t)} &= \mathbf{h}_{(t)} = \mathbf{o}_{(t)} \otimes \tanh\left(\mathbf{c}_{(t)}\right)
\end{aligned}
$$

$b_i, b_f, b_o, b_g$

*Términos de bias para cada una de las cuatro capas*

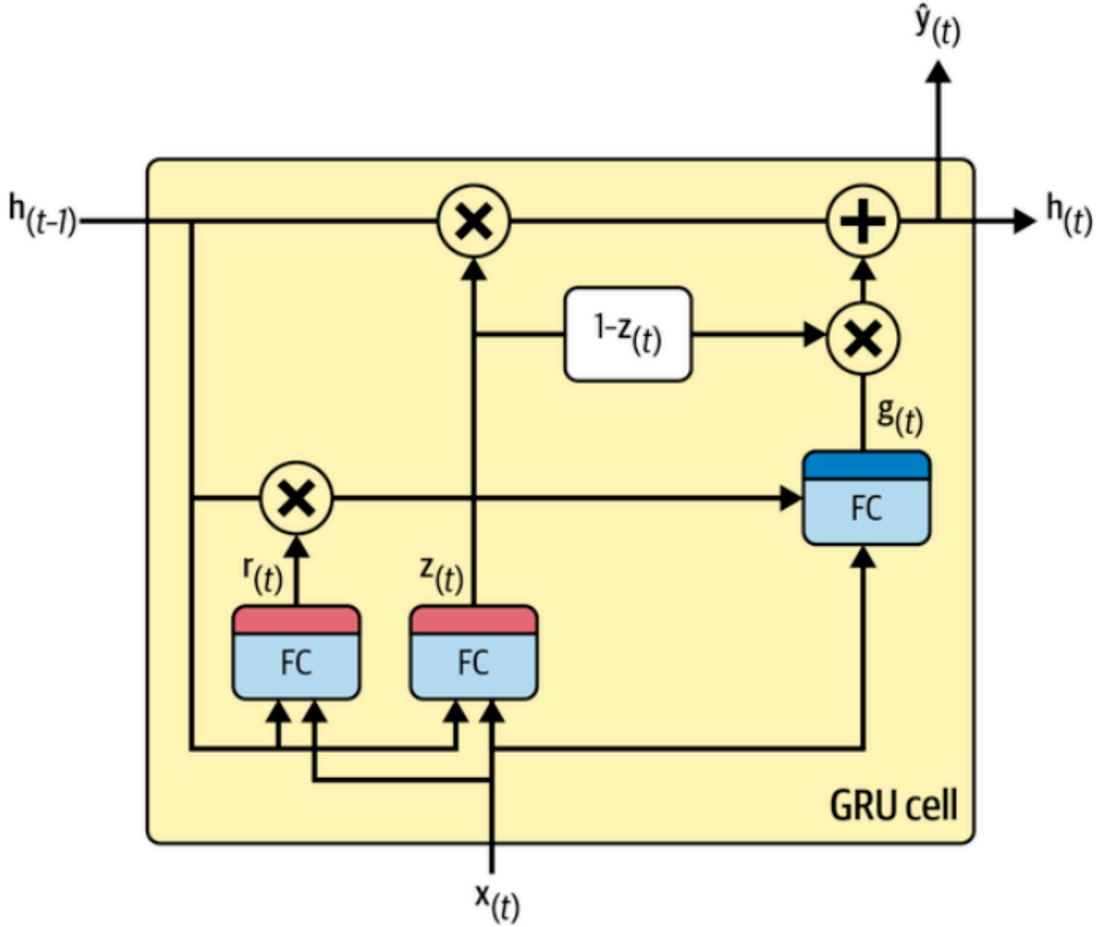
*Inicializa los términos de sesgo en 1, no en 0.*

# RNN, LSTM y GRU

## Gated Recurrent Unit

### GRU cells

*Versión simplificada de una celda LSTM. Y performa igual de bien.*




Kyunghyun Cho et al. in a 2014 paper

# RNN, LSTM y GRU

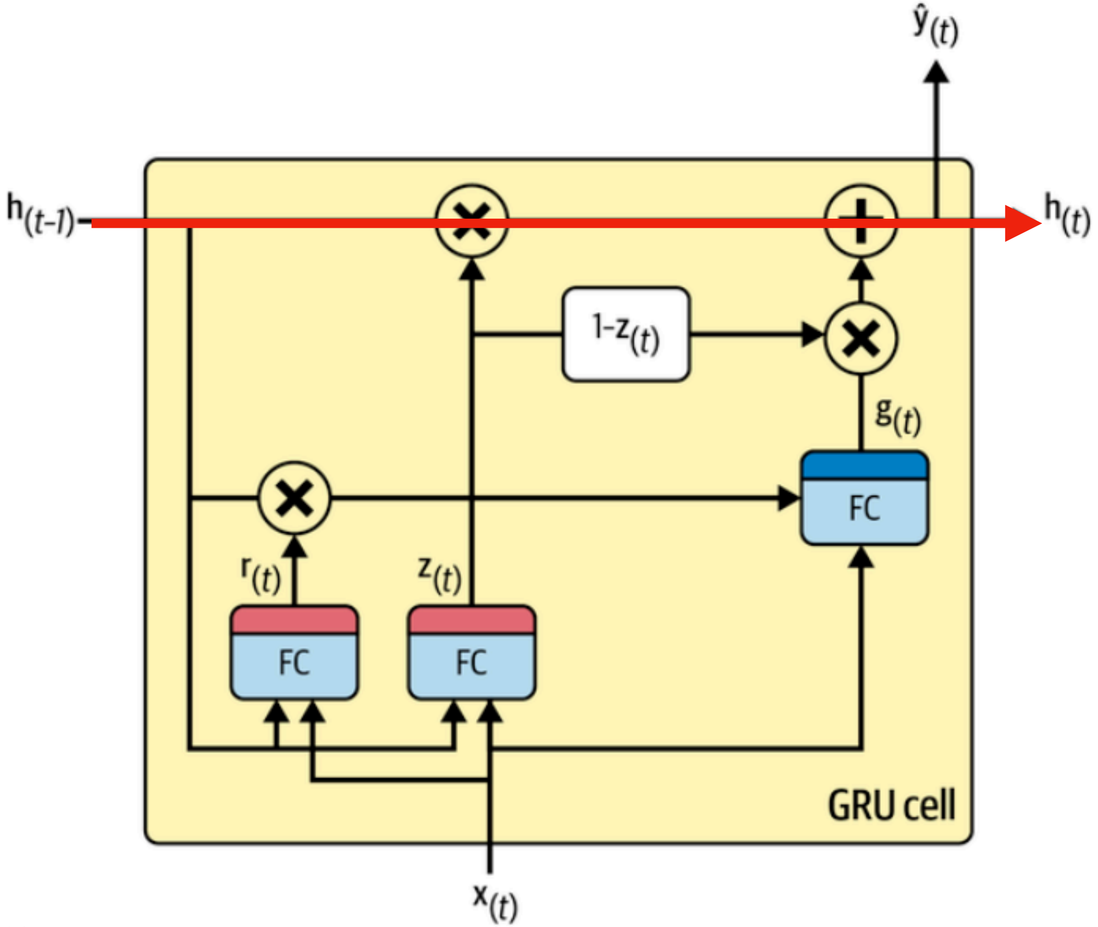
## Gated Recurrent Unit

### GRU cells

*Versión simplificada de una celda LSTM. Y performa igual de bien.*

*Ambos vectores de estado se fusionan en un único vector*

$$h_{(t)}$$




Kyunghyun Cho et al. in a 2014 paper

# RNN, LSTM y GRU

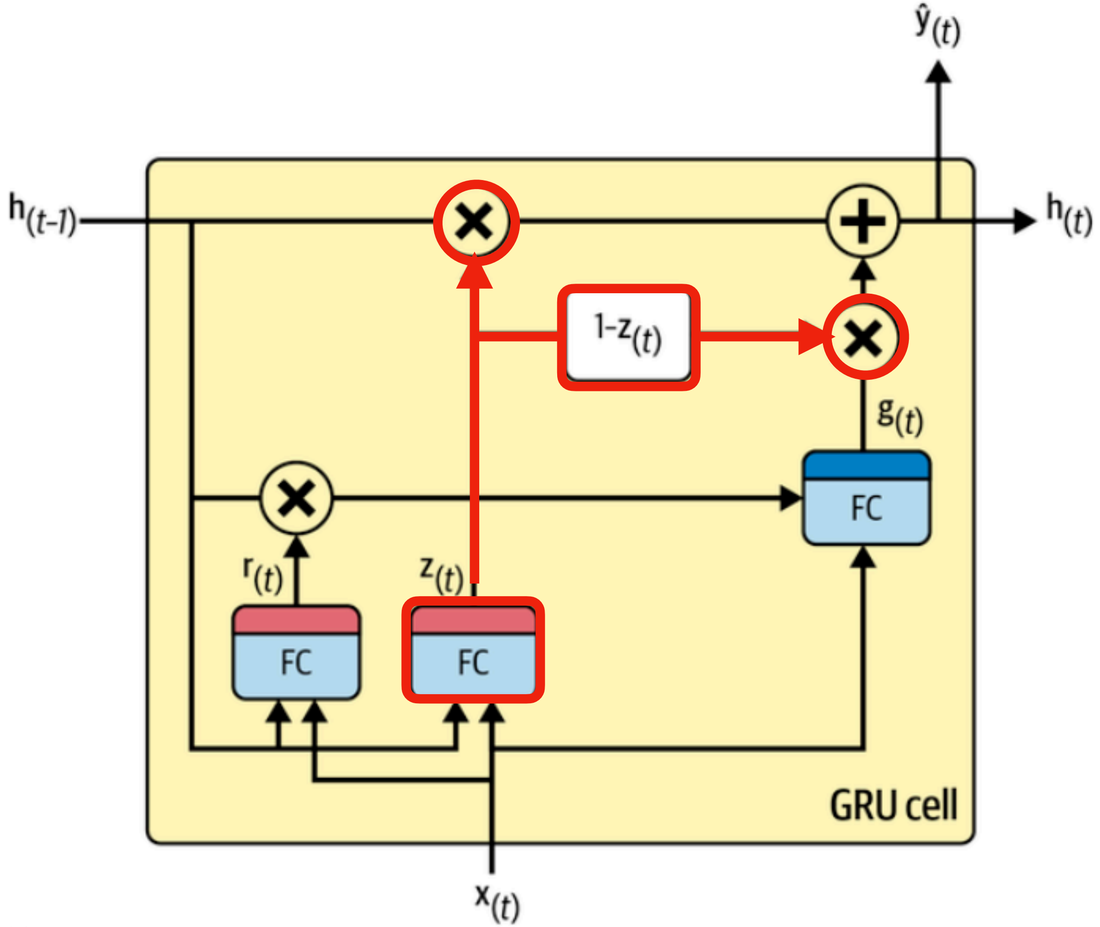
## Gated Recurrent Unit

### GRU cells

*Versión simplificada de una celda LSTM. Y performa igual de bien.*

*Un controlador de puerta única controla tanto la puerta de olvido como la puerta de entrada*

$$z_{(t)}$$




Kyunghyun Cho et al. in a 2014 paper

# RNN, LSTM y GRU

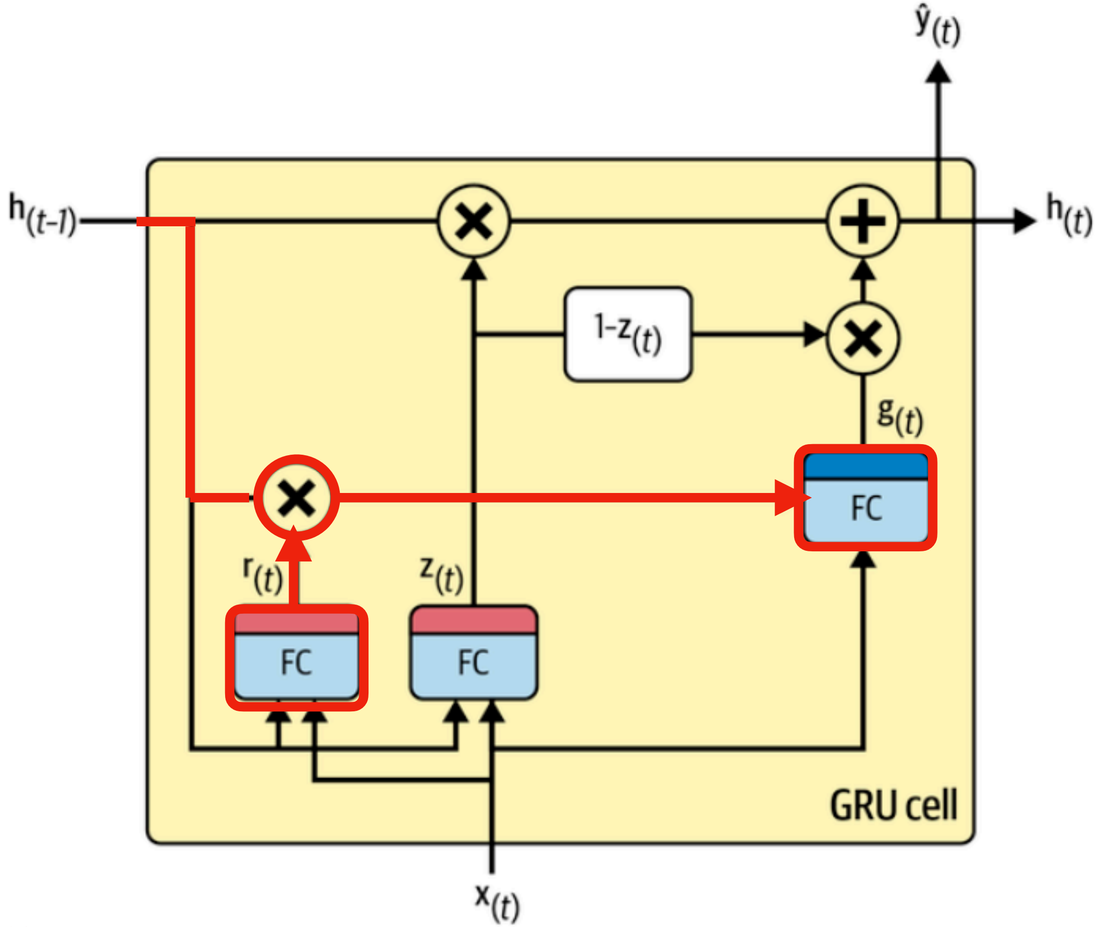
## Gated Recurrent Unit

### GRU cells

*Versión simplificada de una celda LSTM. Y performa igual de bien.*

*Nuevo controlador que determina el porcentaje del estado anterior se mostrará en la capa principal*

$$r_{(t)}$$




Kyunghyun Cho et al. in a 2014 paper

# RNN, LSTM y GRU

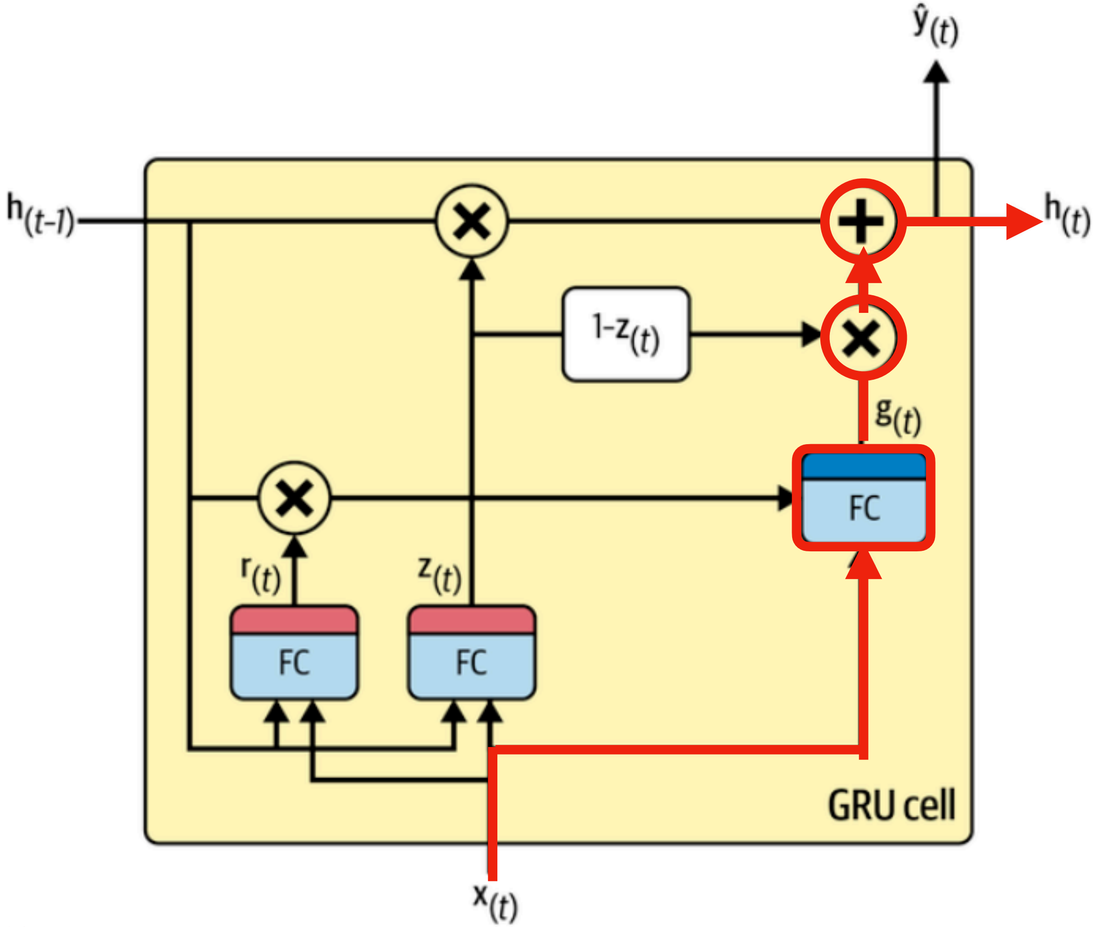
## Gated Recurrent Unit

### GRU cells

*Versión simplificada de una celda LSTM. Y performa igual de bien.*

*Capa principal...*

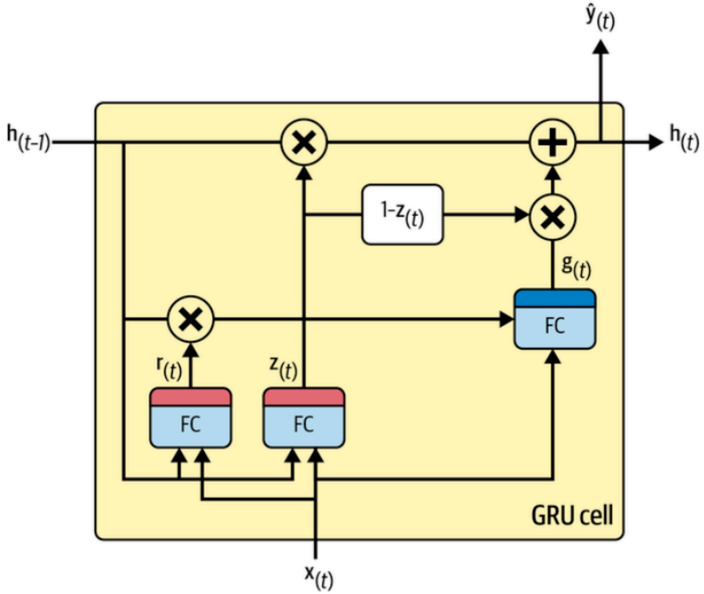
$$g_{(t)}$$




Kyunghyun Cho et al. in a 2014 paper

# RNN, LSTM y GRU

## Gated Recurrent Unit




### GRU cells

*Versión simplificada de una celda LSTM. Y performa igual de bien.*

**Funciones dentro de la celda LSTM**

**Computo**

$$
\begin{aligned}
\mathbf{z}_{(t)} &= \sigma\left(\mathbf{W}_{xz}{}^{\intercal}\mathbf{x}_{(t)} + \mathbf{W}_{hz}{}^{\intercal}\mathbf{h}_{(t-1)} + \mathbf{b}_z\right) \\
\mathbf{r}_{(t)} &= \sigma\left(\mathbf{W}_{xr}{}^{\intercal}\mathbf{x}_{(t)} + \mathbf{W}_{hr}{}^{\intercal}\mathbf{h}_{(t-1)} + \mathbf{b}_r\right) \\
\mathbf{g}_{(t)} &= \tanh\left(\mathbf{W}_{xg}{}^{\intercal}\mathbf{x}_{(t)} + \mathbf{W}_{hg}{}^{\intercal}\left(\mathbf{r}_{(t)} \otimes \mathbf{h}_{(t-1)}\right) + \mathbf{b}_g\right) \\
\mathbf{h}_{(t)} &= \mathbf{z}_{(t)} \otimes \mathbf{h}_{(t-1)} + \left(1 - \mathbf{z}_{(t)}\right) \otimes \mathbf{g}_{(t)}
\end{aligned}
$$

Kyunghyun Cho et al. in a 2014 paper