Clase 23 IMA357

Mg. Alejandro Ferreira Vergara

June 19, 2023

1 Red Neuronal Recurrente (RNN) y Modelo LSTM

- **Arquitectura RNN y LSTM**

1.1 Datos Secuenciales

Lo que hace que las secuencias hayan que tratarlas de formas únicas, respecto a otros tipos de datos, es que **los elementos de una secuencia aparecen en un orden determinado**, y no son independientes unos de otros. **El orden sí importa**.

Representaremos las secuencias como $\left(x^{(1)},\ x^{(2)}, ...,\ x^{(T)}\right)$.

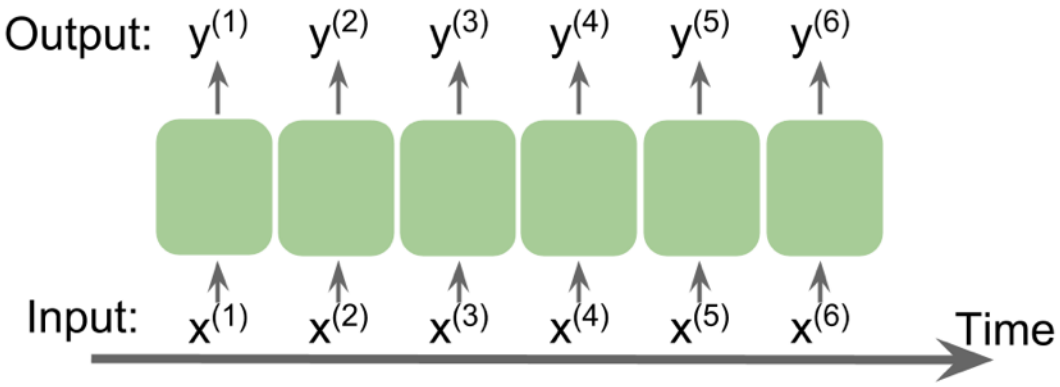
Los superíndice indican el orden de las instancias en la secuencia. T es la longitud de la secuencia.

Un ejemplo particular de secuencias son los datos de **series temporales o series de tiempo**, donde cada punto de muestra $x^{(t)}$ pertenece a un tiempo particular t.

[1]:
```
from IPython.display import Image

Image(filename=r'clase23/16_01.png', width=500)
```

[1]:



1.1.1 Categorias en el Modelado de Secuencias

Si la entrada o la salida son una secuencia, los datos formarán una de las siguientes tres categorías:

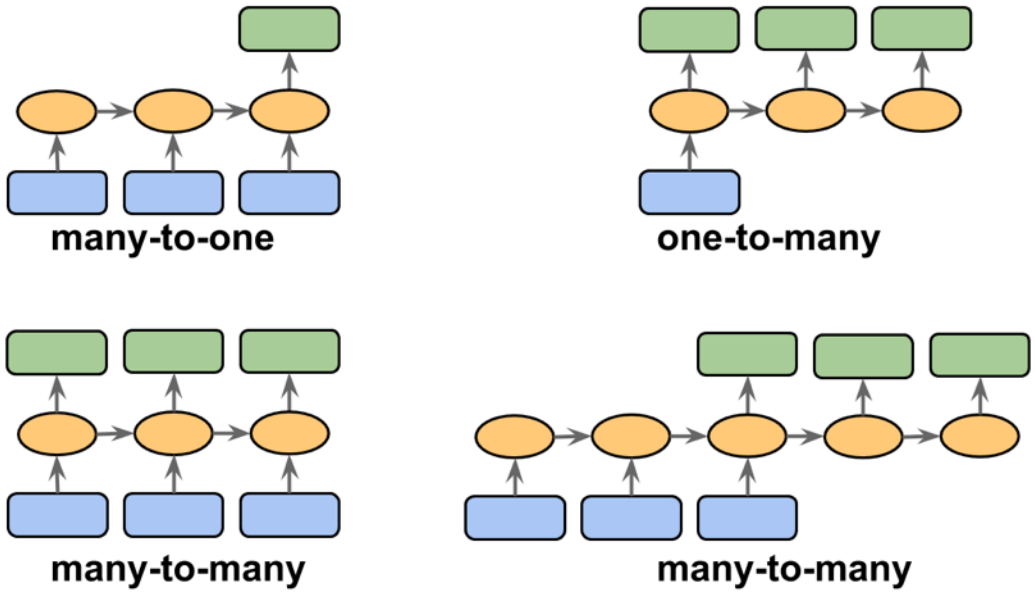
1) **De muchos a uno (Many-to-one)**: Los datos de entrada son una secuencia, pero la salida es un vector de tamaño fijo, no una secuencia.

2) **Uno a muchos (One-to-many)**: Los datos de entrada tienen un formato estándar, no una secuencia, pero la salida es una secuencia.

3) **De muchos a muchos (Many-to-many)**: Tanto la matriz de entrada como la de salida son secuencias. Esta categoría puede dividirse en función de si la entrada y la salida están sincronizadas o no. Un ejemplo de tarea de modelado **sincronizado (synchronized)** de muchos a muchos es la clasificación de vídeos, en la que se etiqueta cada fotograma de un vídeo. Un ejemplo de tarea de muchos a muchos con **retraso (delayed)** sería la traducción de un idioma a otro. Por ejemplo, una frase entera en inglés debe ser leída y procesada por una máquina antes de producir su traducción al español.

[2]:
```
Image(filename=r'clase23/16_02.png', width=600)
```

[2]:



1.2 RNNs para Modelar Secuencias

En una red estándar feedforward, la información fluye de la entrada a la capa oculta, y luego de la capa oculta a la capa de salida.

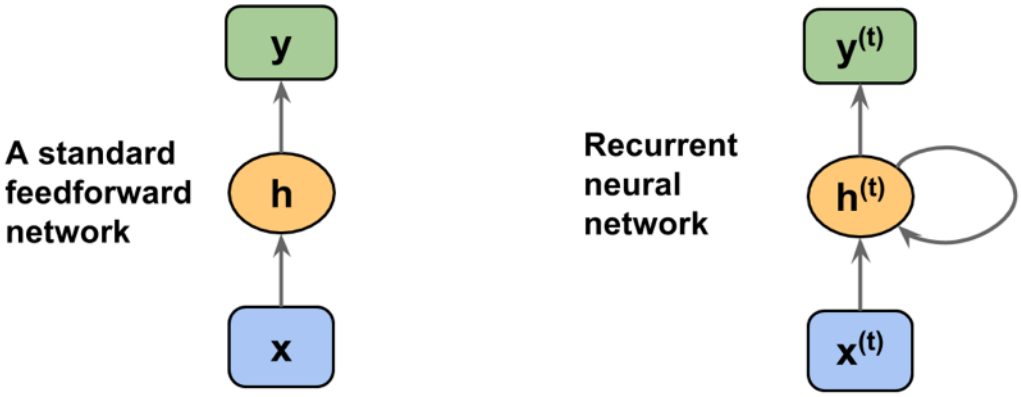
En cambio, en una Red Recurrente, la capa oculta recibe su entrada tanto de la capa de entrada como de la capa oculta en el paso de tiempo anterior.

El flujo de información en pasos de tiempo adyacentes en la capa oculta **permite a la red tener una memoria de eventos pasados.**

Este flujo de información suele mostrarse como un bucle, también conocido como **arista recurrente** en la notación gráfica, que es como esta arquitectura general obtuvo su nombre.

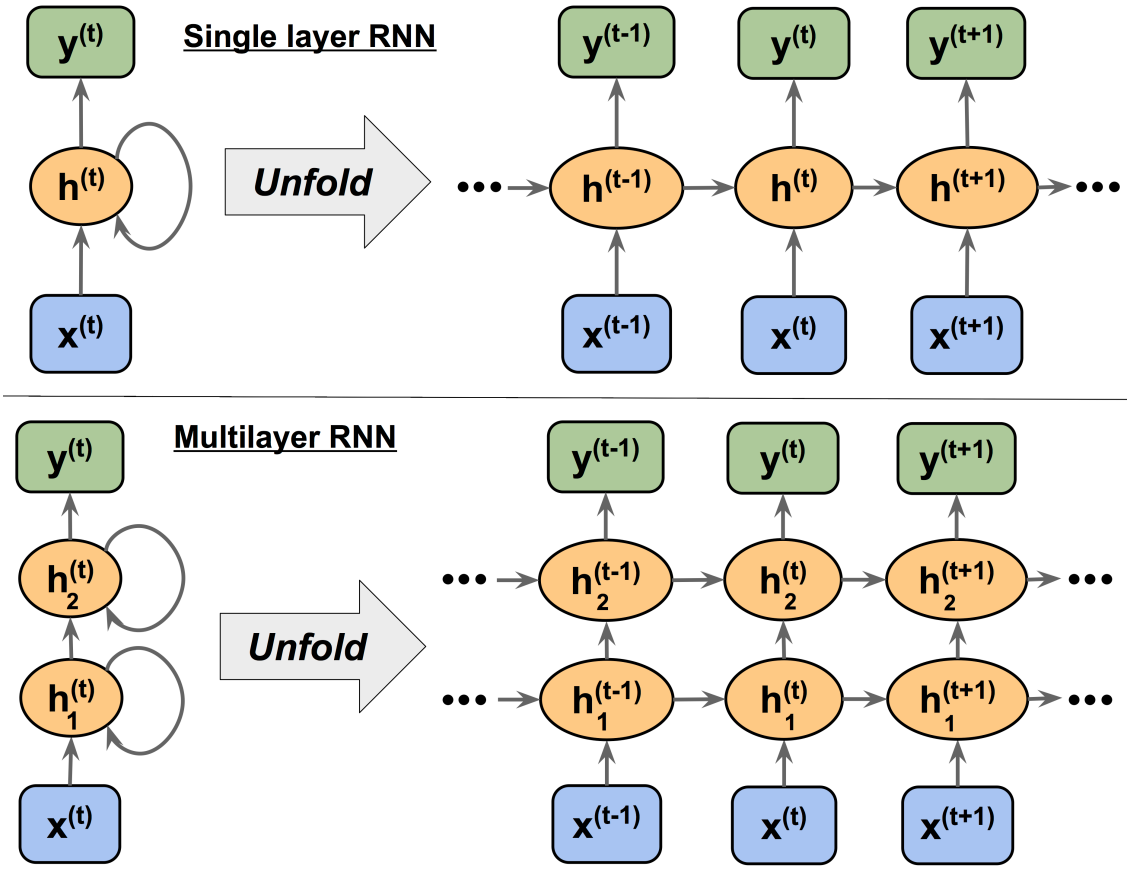
[3]:
```
Image(filename=r'clase23/16_03.png', width=600)
```

[3]:



[4]:
```
Image(filename=r'clase23/16_04.png', width=550)
```

[4]:



En el primer paso de tiempo $t = 0$, las unidades ocultas se inicializan con ceros o pequeños valores aleatorios. Luego, en un paso de tiempo donde $t > 0$, las unidades ocultas obtienen su entrada del

punto de datos en el tiempo actual $x^{(t)}$ y los valores anteriores de las unidades ocultas en $t-1$, indicados como $h^{(t-1)}$.

Del mismo modo, en el caso de una **RNN Multicapa**, podemos resumir el flujo de información como:

Capa = 1: Aquí, la capa oculta se representa como $h_1^{(t)}$ y obtiene su entrada del punto de datos $x^{(t)}$ y los valores ocultos en la misma capa, pero el paso de tiempo anterior $h_1^{(t-1)}$

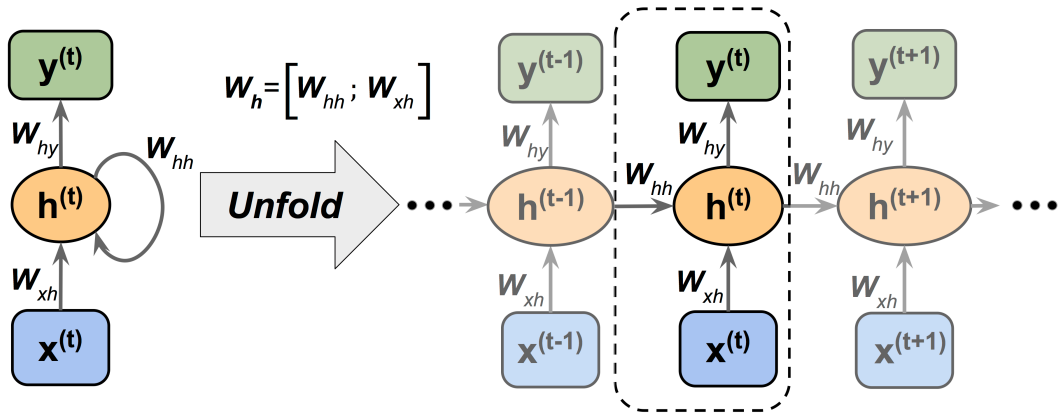
Capa = 2: La segunda capa oculta, $h_2^{(t)}$ recibe sus entradas de las unidades ocultas de la capa inferior en el paso de tiempo actual $h_1^{(t)}$ y sus propios valores ocultos del paso de tiempo anterior $h_2^{(t-1)}$

1.2.1 Cálculo de las activaciones en una RNN

Para simplificar, consideraremos una sola capa oculta; sin embargo, el mismo concepto se aplica a las RNNs multicapa.

[5]:
```
Image(filename=r'clase23/16_05.png', width=600)
```

[5]:



Cada arista dirigida (las conexiones entre cajas en la figura anterior) en la representación de una RNN está asociada a una matriz de pesos.

Esos pesos no dependen del tiempo t; por lo tanto, son compartidos a lo largo del eje temporal.

Las diferentes **matrices de pesos en una RNN de una sola capa** son las siguientes:

W_{xh} : La matriz de pesos entre la entrada $x^{(t)}$ y la capa oculta h.

W_{hh} : La matriz de pesos asociada a la arista recurrente.

W_{hy} : La matriz de pesos entre la capa oculta y la capa de salida.

En ciertas implementaciones, se puede observar que las matrices de pesos W_{xh} y W_{hh} se concatenan en una **matriz combinada** $W_h = [W_{xh}; W_{hh}]$**.**

Para la capa oculta, la entrada de la red z_h (preactivación) se **calcula mediante una combinación lineal**. Es decir, calculamos la suma de las multiplicaciones de las matrices de pesos con los vectores correspondientes y añadimos la unidad de sesgo $z_h^{(t)} = W_{xh}x^{(t)} + W_{hh}h^{(t-1)} + b_h$:

$$h^{(t)} = \phi_h\left(z_h^{(t)}\right) = \phi_h\left(W_{xh}x^{(t)} + W_{hh}h^{(t-1)} + b_h\right)$$

b_h : vector de sesgo para las unidades ocultas

$\phi_h(\cdot)$: función de activación de la capa oculta.

En caso de querer utilizar la **matriz de pesos concatenados** $W_h = [W_{xh}; W_{hh}]$, la fórmula para calcular las unidades ocultas cambiará de la siguiente manera:

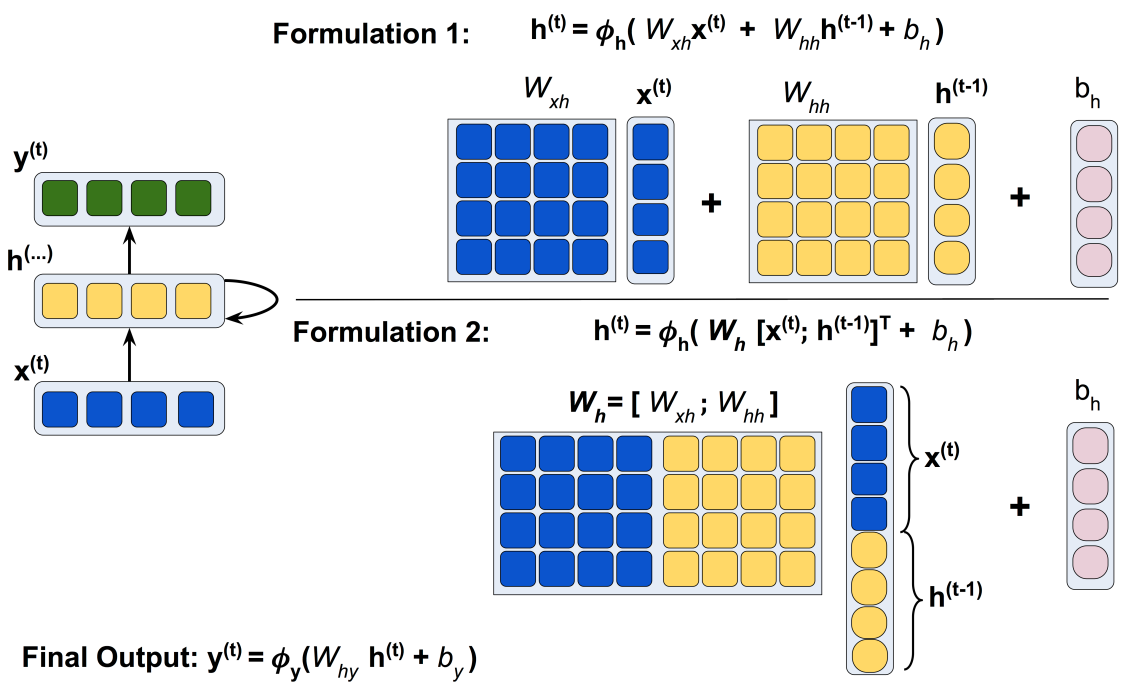
$$h^{(t)} = \phi_h\left([W_{xh}; W_{hh}]\begin{bmatrix} x^{(t)} \\ h^{(t-1)} \end{bmatrix} + b_h\right)$$

Una vez calculadas las activaciones de las unidades ocultas en el paso de tiempo actual, las activaciones de las unidades de salida se calcularán como:

$$y^{(t)} = \phi_y\left(W_{hy}h^{(t)} + b_y\right)$$

[6]:
```
Image(filename=r'clase23/16_06.png', width=600)
```

[6]:



1.3 Entrenamiento de una RNN

El algoritmo de aprendizaje de las RNNs se introdujo en 1990: **Retropropagación a través del tiempo (BPTT).**

La idea básica es que la pérdida global L en una RNN, es la suma de todas las funciones de pérdida en los tiempos $t = 1$ a $t = T$:

$$L = \sum_{t=1}^{T} L^{(t)}$$

Dado que la pérdida en el momento $1 : t$ depende de las unidades ocultas en todos los pasos de tiempo anteriores a $1 : t$, el gradiente se calculará como:

$$\frac{\partial L^{(t)}}{\partial W_{hh}} = \frac{\partial L^{(t)}}{\partial y^{(t)}} \times \frac{\partial y^{(t)}}{\partial h^{(t)}} \times \left(\sum_{k=1}^{t} \frac{\partial h^{(t)}}{\partial h^{(k)}} \times \frac{\partial h^{(k)}}{\partial W_{hh}} \right)$$

Aquí $\frac{\partial h^{(t)}}{\partial h^{(k)}}$,se calcula como una multiplicación de pasos de tiempo adyacentes:

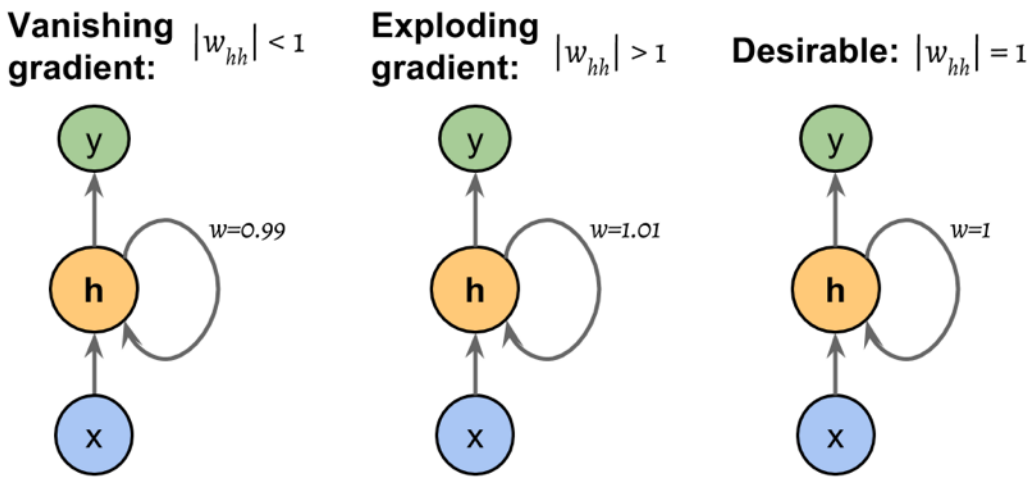
$$\frac{\partial h^{(t)}}{\partial h^{(k)}} = \prod_{i=k+1}^{t} \frac{\partial h^{(i)}}{\partial h^{(i-1)}}$$

La retropropagación a través del tiempo, o BPTT, introduce algunos retos nuevos.

Debido al factor multiplicativo $\frac{\partial h^{(t)}}{\partial h^{(k)}}$ en el cálculo de los gradientes de una función de pérdida, surge el llamado problema del gradiente **desvanecido** o **explosivo** (vanishing or exploding).

[7]:
```
Image(filename=r'clase23/16_07.png', width=600)
```

[7]:



En la práctica, hay dos soluciones a este problema:

- La retropropagación truncada en el tiempo (Truncated backpropagation through time) (TBPTT), que recorta los gradientes por encima de un umbral determinado. Aunque la TBPTT puede resolver el problema de la explosión del gradiente, el truncamiento limita el número de pasos en los que el gradiente puede fluir efectivamente hacia atrás y actualizar adecuadamente los pesos.
- Memoria a corto plazo (Long short-term memory) (LSTM), diseñadas en 1997. Han tenido más éxito en el modelado de secuencias de largo alcance al superar el problema del gradiente desvanecido.

1.4 Unidades LSTM

Las **LSTM (Long Short-Term Memory)** se desarrollaron para superar el problema de explosión de gradiente.

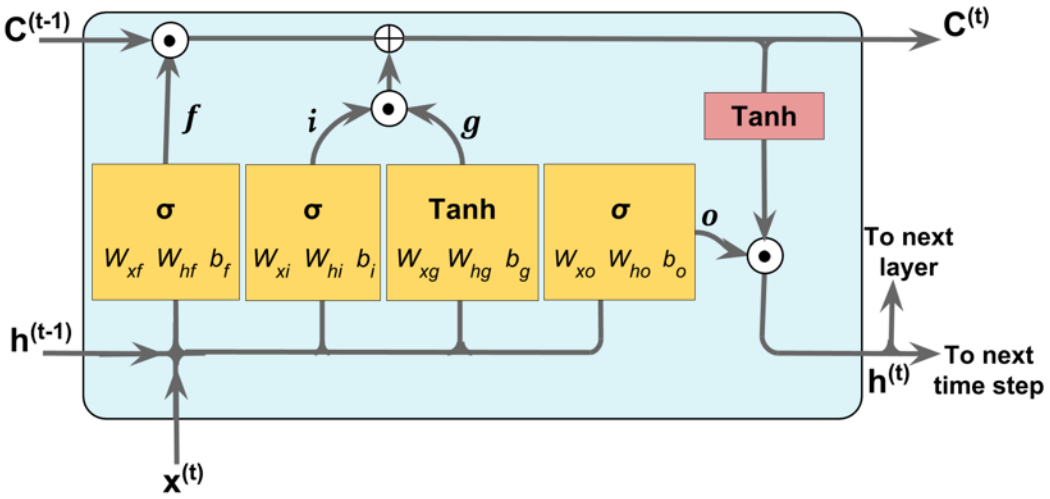
Al bloque de construcción de una LSTM se le llama **celda de memoria**, que representa esencialmente la capa oculta.

En cada celda de memoria, hay una arista recurrente que tiene el peso deseable $w = 1$ para superar los problemas de gradiente de fuga y explosión.

Los valores asociados a esta arista recurrente se denominan **estado de la celda**.

[8]:
```
Image(filename=r'clase23/16_08.png', width=600)
```

[8]:



En la figura anterior, $\odot$ se refiere al **producto elemento-elemento** y $\oplus$ significa **suma elemento-elemento**. Además, $x^{(t)}$ se refiere a los datos de entrada en el tiempo t, y $h^{(t-1)}$ indica las unidades ocultas en el tiempo $t-1$.

Se indican cuatro casillas con una función de activación definida, ya sea la función sigmoide (σ) o la tangente hiperbólica (Tanh).

Un conjunto de pesos; estas casillas aplican la combinación lineal realizando multiplicaciones matriciales-vectoriales en su entrada.

Las unidades de cómputo cuyas unidades de salida pasan por $\odot$, se denominan **puertas**. En una célula LSTM, hay tres tipos de puertas diferentes:

1) La **puerta del olvido (forget gate)** (f_t) permite a la célula de memoria restablecer el estado de la célula sin crecer indefinidamente. De hecho, la puerta del olvido decide qué información se deja pasar y qué información se suprime.

$$f_t = \sigma\left(W_{xf}x^{(t)} + W_{hf}h^{(t-1)} + b_f\right)$$

2) La **puerta de entrada (input gate)** (i_t) y el **nodo de entrada** (g_t) se encargan de actualizar el estado de la célula. Se calculan como:

$$i_t = \sigma\left(W_{xi}x^{(t)} + W_{hi}h^{(t-1)} + b_i\right)$$

$$g_t = tanh\left(W_{xg}x^{(t)} + W_{hg}h^{(t-1)} + b_g\right)$$

El estado de la célula en el momento t se calcula como:

$$C^{(t)} = \left(C^{(t-1)} \odot f_t\right) \oplus \left(i_t \odot g_t\right)$$

3) La **puerta de salida (output gate)** (o_t) decide cómo actualizar los valores de las unidades ocultas:

$$o_t = \sigma\left(W_{xo}x^{(t)} + W_{ho}h^{(t-1)} + b_o\right)$$

Dado esto, las unidades ocultas en el paso de tiempo actual se calculan como:

$$h^{(t)} = o_t \odot tanh\left(C^{(t)}\right)$$

Observar que el estado de la célula del paso de tiempo anterior, $C^{(t-1)}$, se modifica para obtener el estado de la célula en el paso de tiempo actual, $C^{(t)}$, sin que se multiplique directamente con ningún parámetro de peso.

[ ]: