

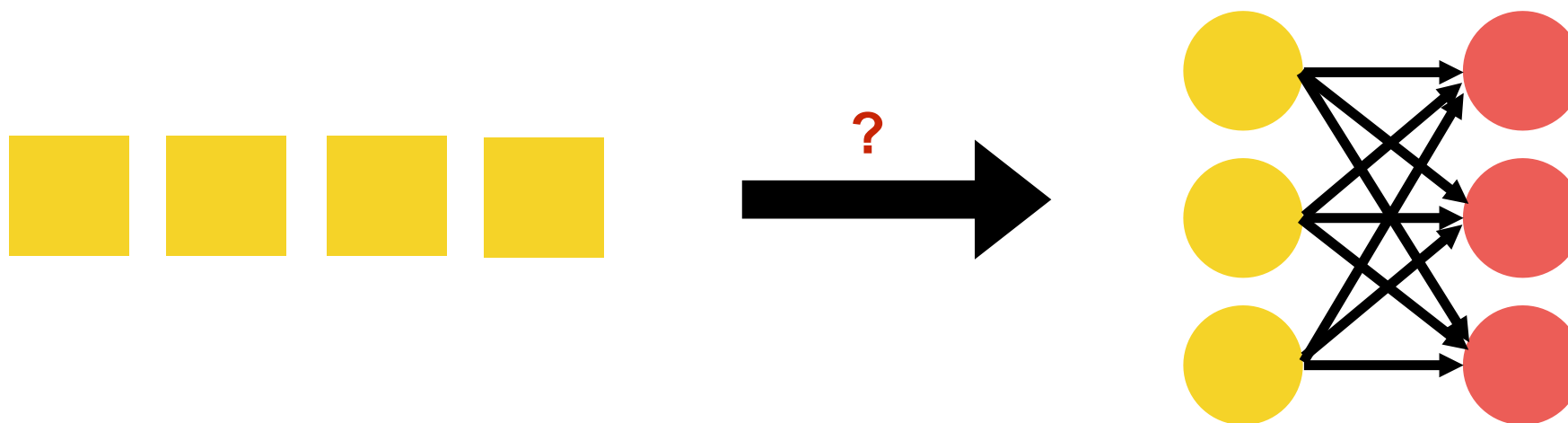
# RNN - Deep Learning Python



# Redes Neuronales Recurrentes

## Recurrent Neural Network (RNN)

- ¿Cómo usar las redes neuronales para datos secuenciales tipo **Series de Tiempo**?
- Cuando el tamaño de la entrada y salida nos son fijos.
- Ejemplos: audio, texto, series temporales financieras...

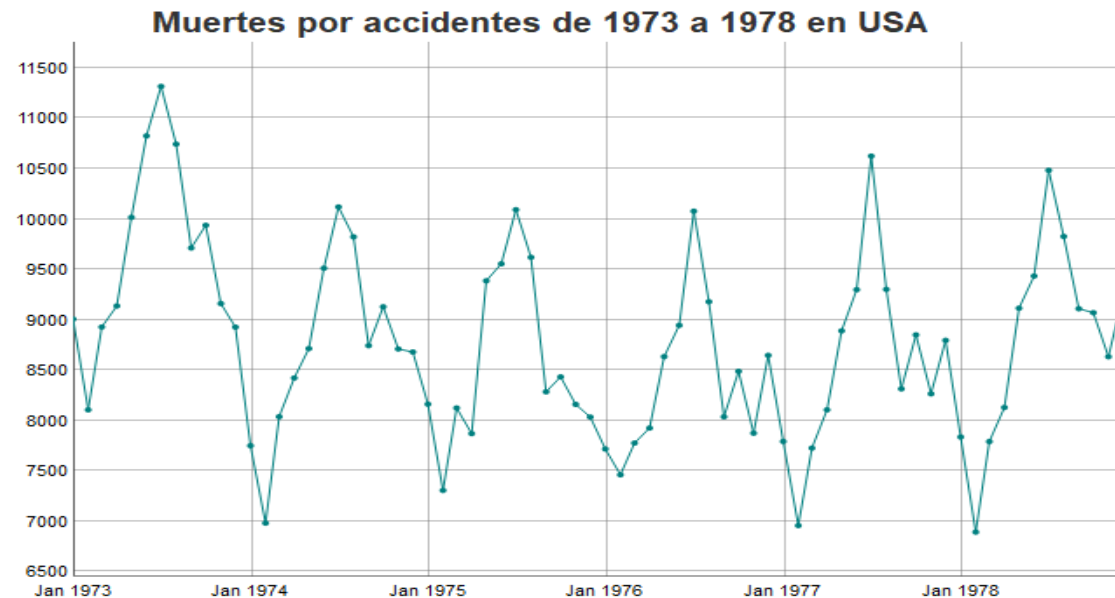


# Pero ¿Qué es una Series de Tiempo?

- Se llama Serie de Tiempo a un conjunto de observaciones sobre valores que toma una variable (cuantitativa) en diferentes momentos del tiempo.
- Una Serie de Tiempo es una colección o conjunto de mediciones de cierto fenómeno o experimento registrados secuencialmente en el tiempo, en forma equiespaciada (a intervalos de tiempo iguales).

# Pero ¿Qué es una Series de Tiempo?

	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
1973	9007	8106	8928	9137	10017	10826	11317	10744	9713	9938	9161	8927
1974	7750	6981	8038	8422	8714	9512	10120	9823	8743	9129	8710	8680
1975	8162	7306	8124	7870	9387	9556	10093	9620	8285	8433	8160	8034
1976	7717	7461	7776	7925	8634	8945	10078	9179	8037	8488	7874	8647
1977	7792	6957	7726	8106	8890	9299	10625	9302	8314	8850	8265	8796
1978	7836	6892	7791	8129	9115	9434	10484	9827	9110	9070	8633	9240



# Ejemplos de Series de Tiempo

- Economía: Precios de un artículo, tasas de desempleo, tasa de inflación, índice de precios, precio del dólar, precio del cobre, precios de acciones, ingreso nacional bruto.
- Meteorología: Cantidad de agua caída, temperatura máxima diaria, Velocidad del viento (energía eólica), energía solar.
- Geofísica: Series sismológicas.

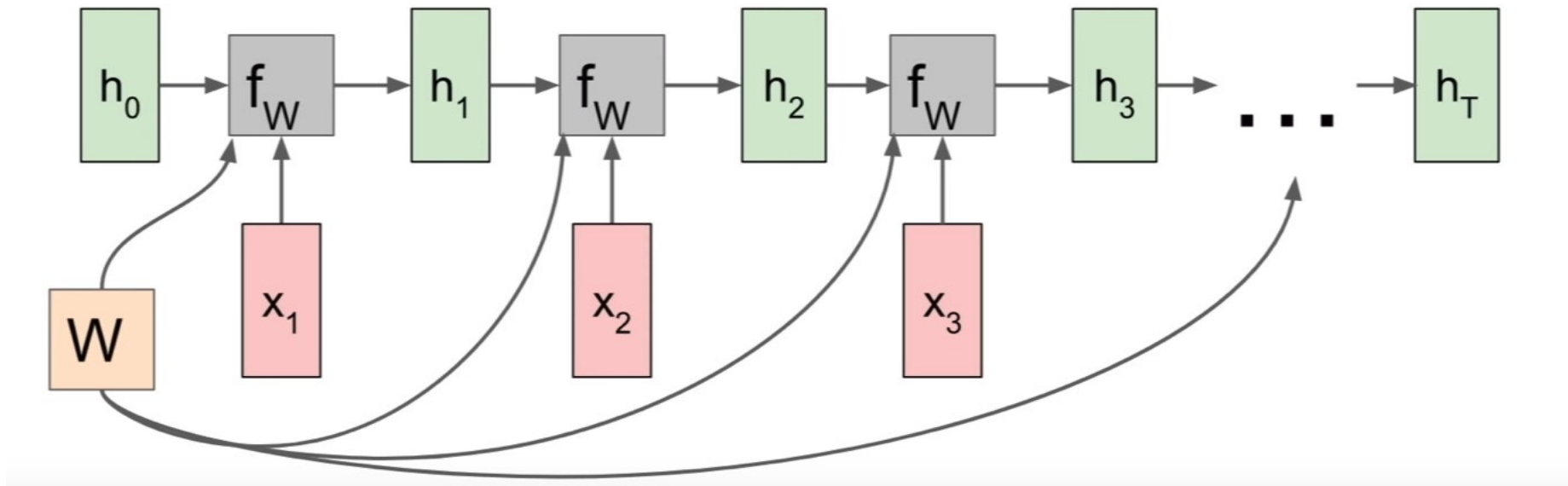
# Ejemplos de Series de Tiempo

- Química: Viscosidad de un proceso, temperatura de un proceso.
- Demografía: Tasas de natalidad, tasas de mortalidad.
- Medicina: Electrocardiograma, electroencefalograma.
- Marketing: Series demanda, gastos, utilidades, ventas, ofertas.

# Redes Neuronales Recurrentes

## Series de Tiempo

Vista gráfica de la secuencia computacional de los cálculos:



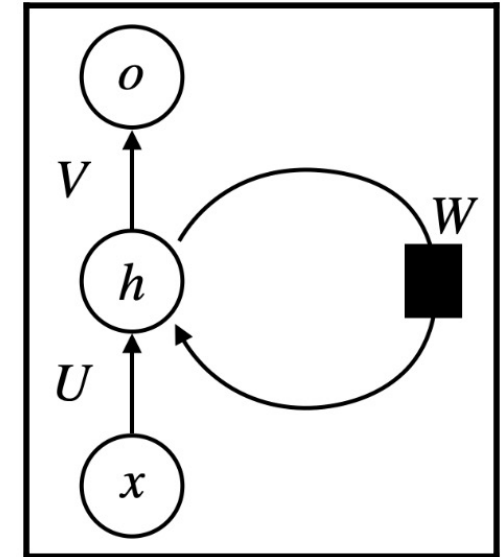


# Redes Neuronales Recurrentes

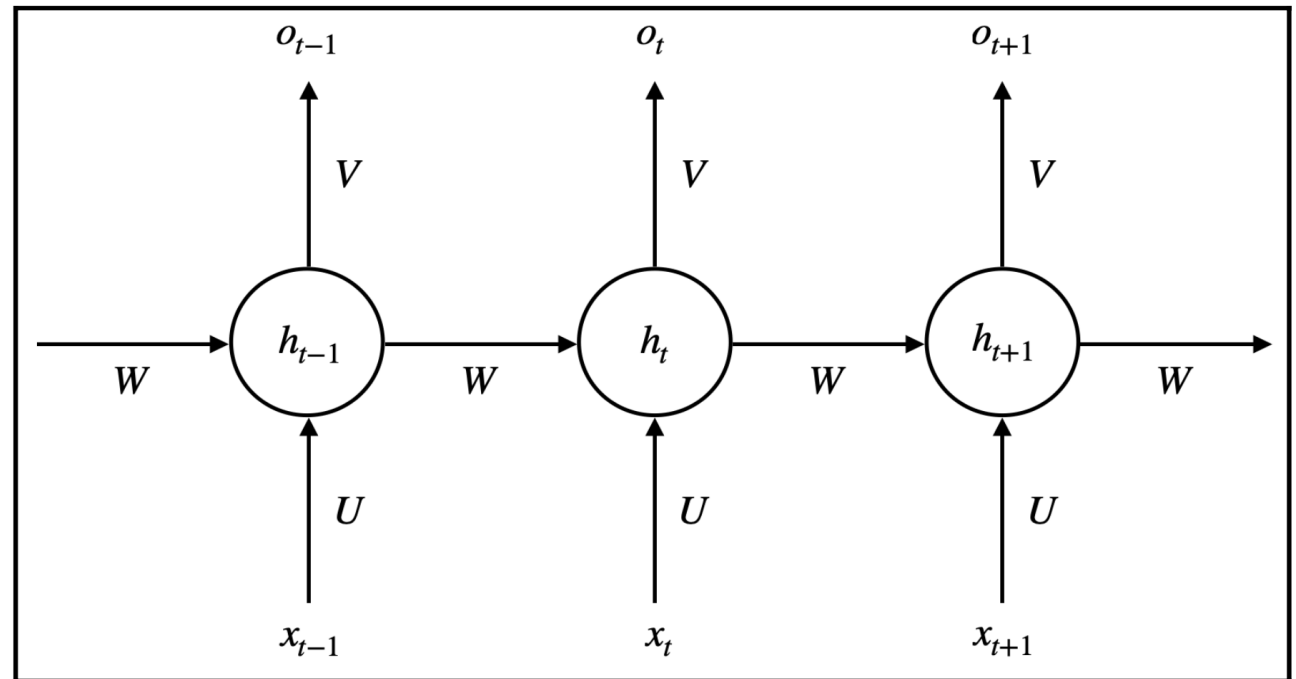
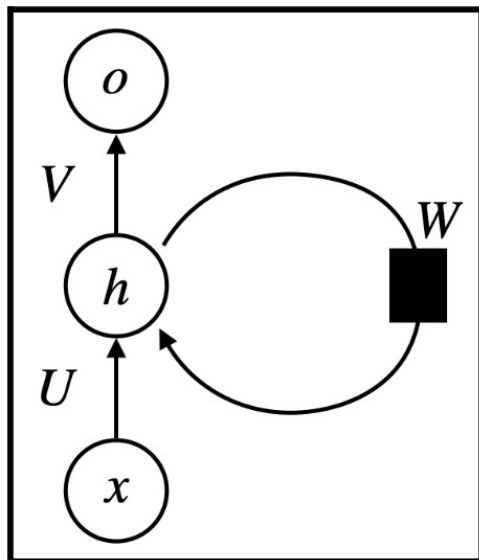
- Las RNN (Recurrent Neural Network) son muy efectivas para **datos secuenciales** y se utilizan en aplicaciones como comercio, subtítulos de imágenes, clasificación de sentimientos, traducción de idiomas, clasificación de videos, entre otros.
- En las Redes Neuronales Recurrentes, se asume que todas las entradas y salidas son dependientes, es decir, ***cada salida depende de la anterior***, lo que les permite capturar dependencias en secuencias, como en el **lenguaje**, donde la siguiente palabra depende de la palabra anterior y la anterior y así sucesivamente.
- La palabra **Recurrente** en el nombre de esta red neuronal proviene del hecho de que tiene conexiones *cíclicas* (recursivas) y se realiza el mismo cálculo en cada elemento de la secuencia. Esto le permite aprender (o memorizar) partes de los datos para hacer predicciones sobre el futuro. La ventaja de un RNN es que puede escalar a secuencias mucho más largas que los modelos que no se basan en secuencias.

## Las RNN pueden ser inicialmente más difíciles de entender en comparación con las MLP o las CNN.

- En un MLP, el **perceptrón** es la unidad fundamental. Un MLP es solo una *red de perceptrones*.
- En una CNN, el **kernel** es una ventana que se desliza a través del **mapa de características** para generar otro mapa de características.
- En un RNN, lo más importante es el concepto de **bucle/ciclo** automático. De hecho, podría haber solo una celda.



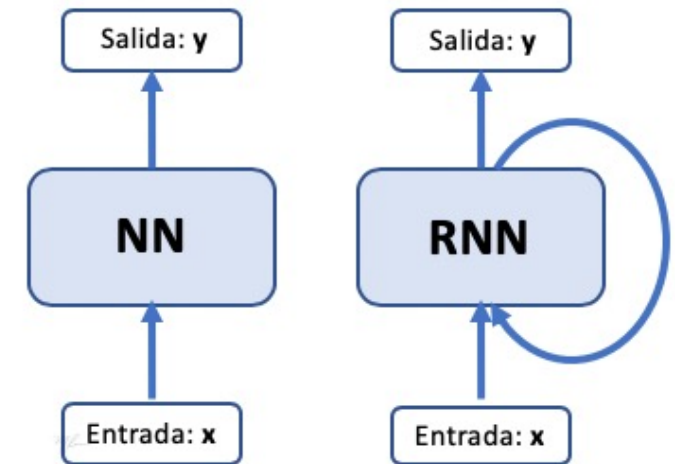
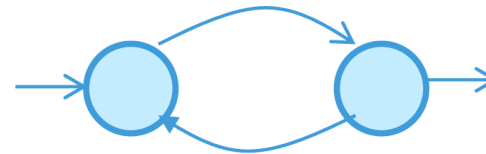
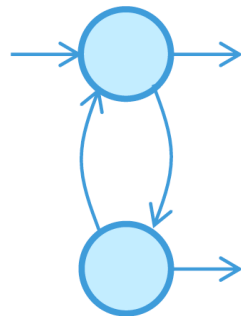
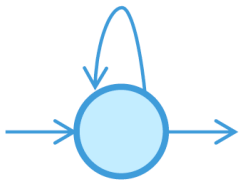
- La ilusión de múltiples celdas aparece porque una celda existe en el paso de tiempo, pero en realidad es la misma celda que se reutiliza repetidamente a menos que se "desenrolle" la red. Las redes neuronales subyacentes de los RNN comparten un nodo el celda.



# Recurrencia

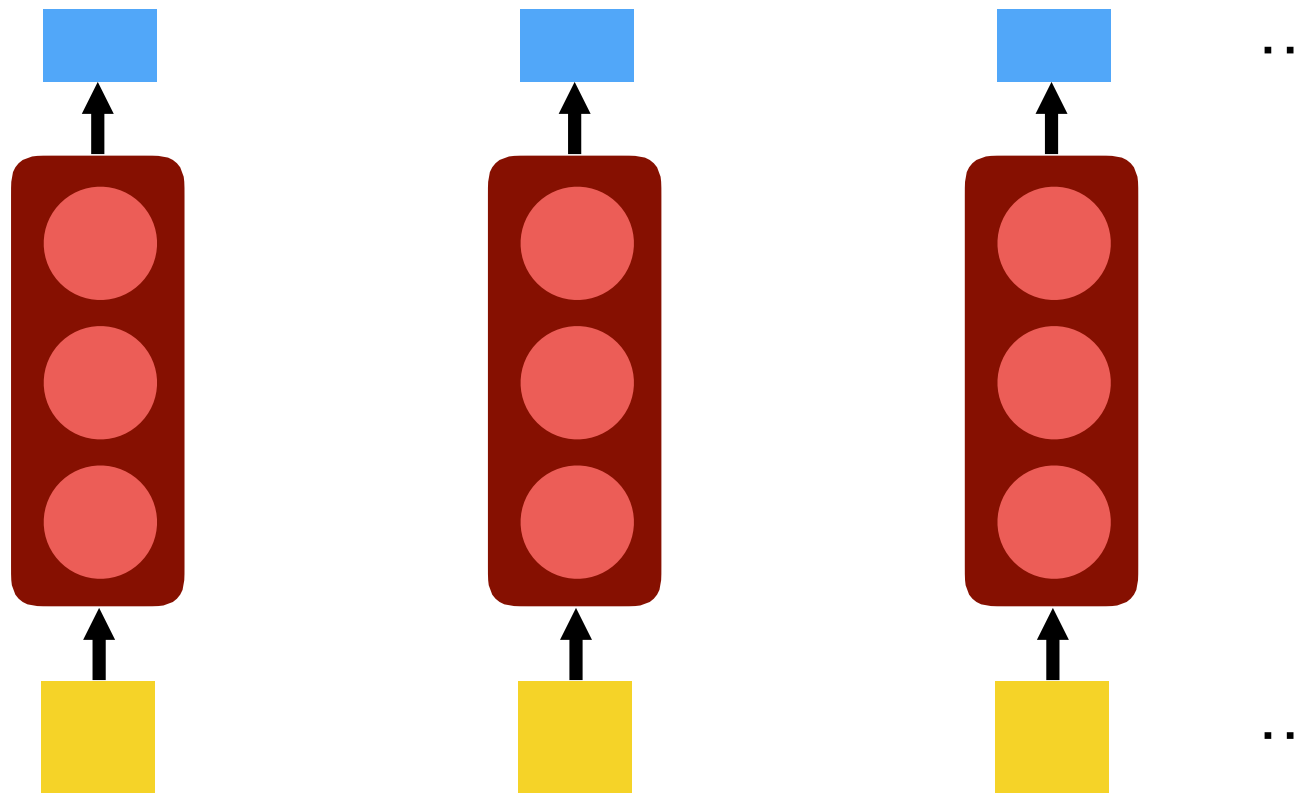
Pueden tener ciclos o bucles en las conexiones (conexiones recurrentes).  
Las conexiones recurrentes pueden ser:

1. De una neurona con ella misma.
2. Entre neuronas de una misma capa.
3. Entre neuronas de una capa a una capa anterior.

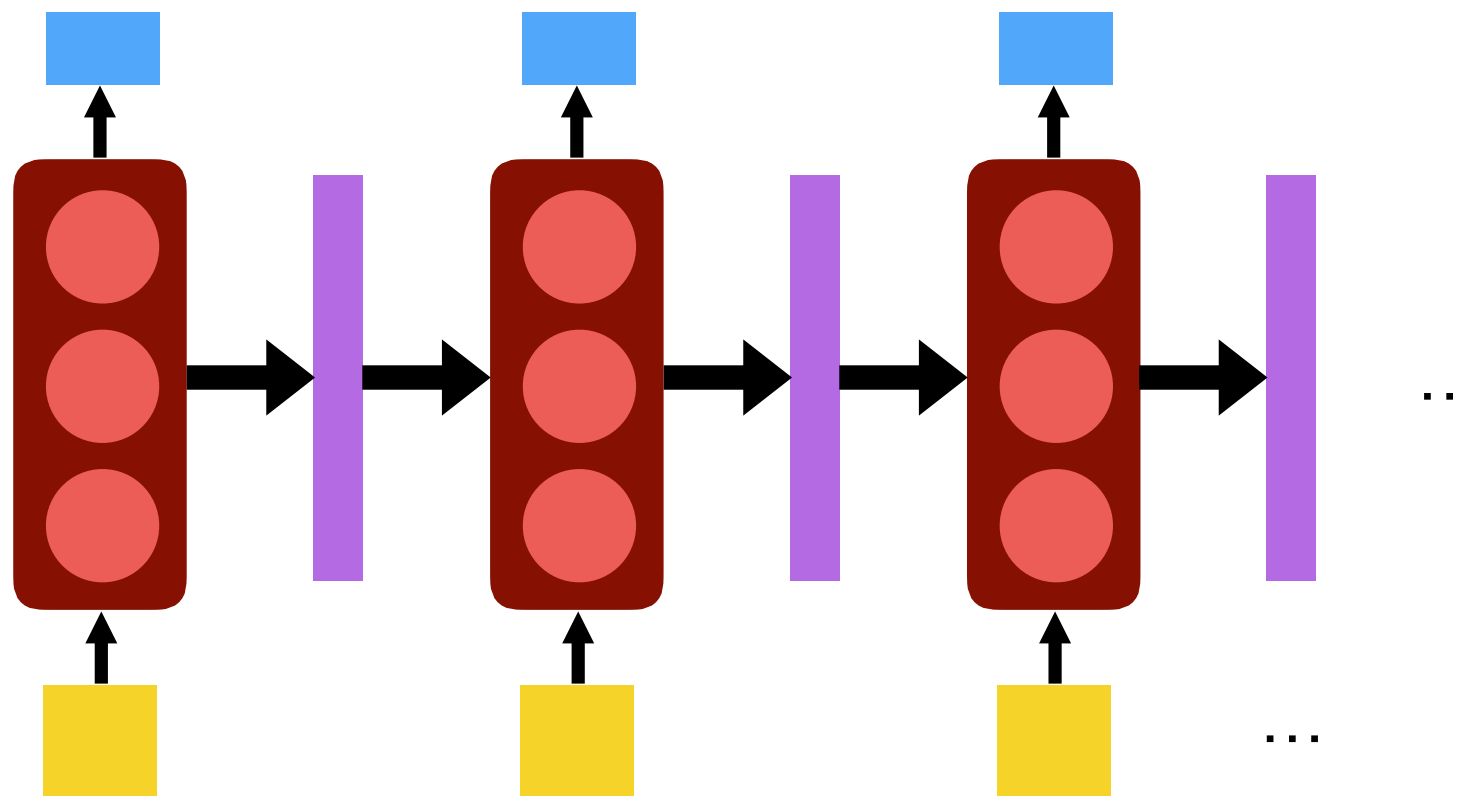


# Redes Neuronales Recurrentes

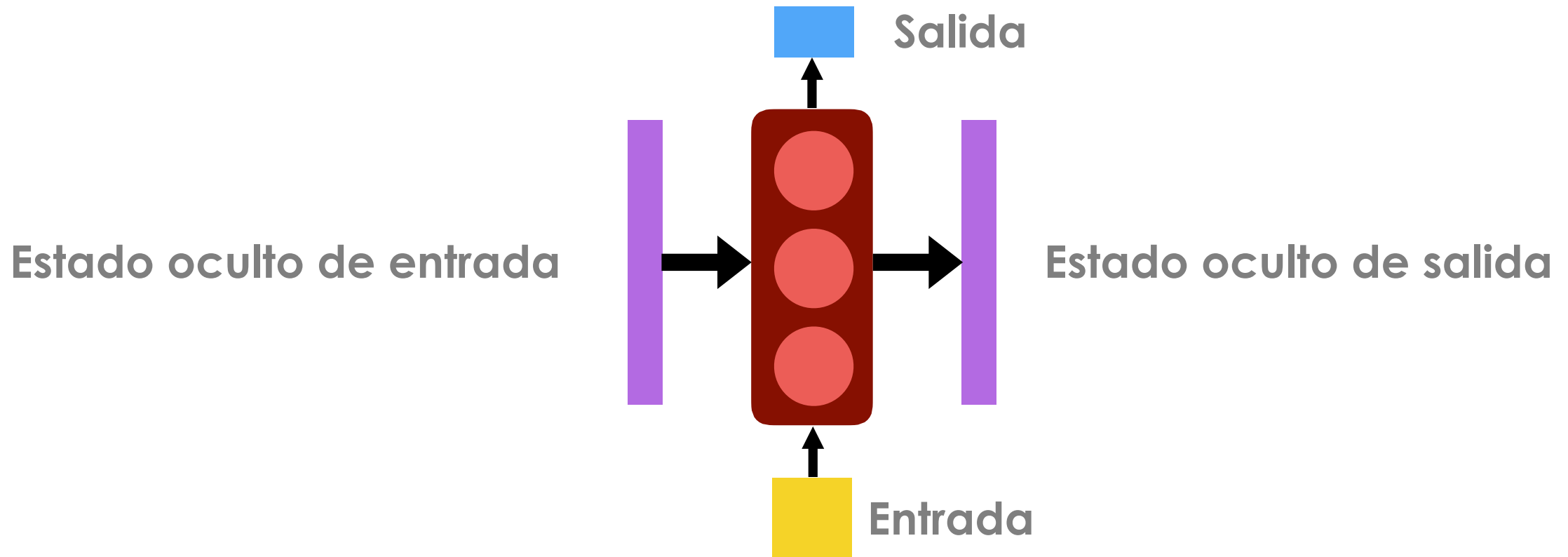
Se usan los ***mismos pesos*** para cada paso de tiempo.



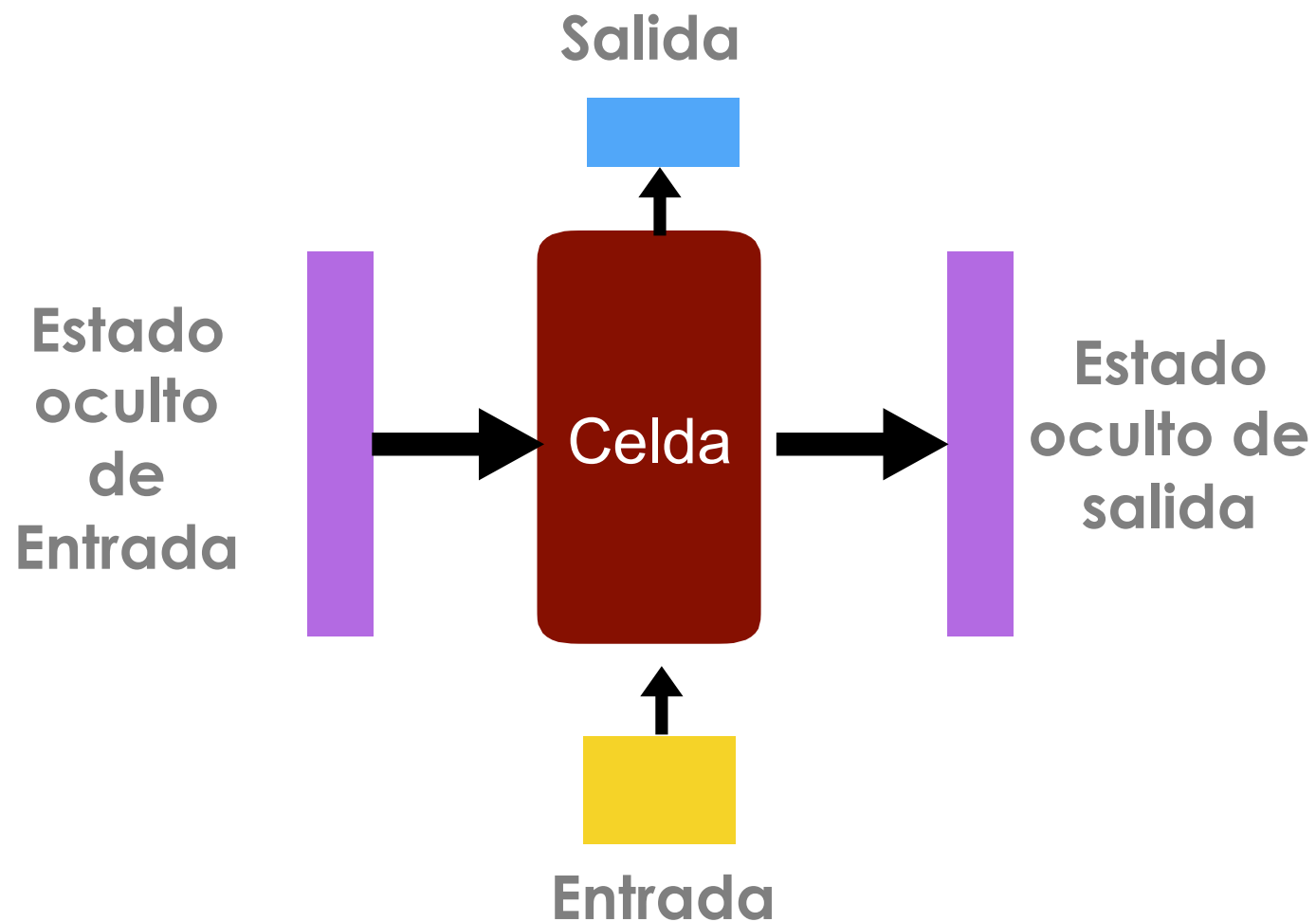
La información se pasa a través de pasos de tiempo usando estados ocultos ...



# Estados ocultos



# Tipos de Celdas

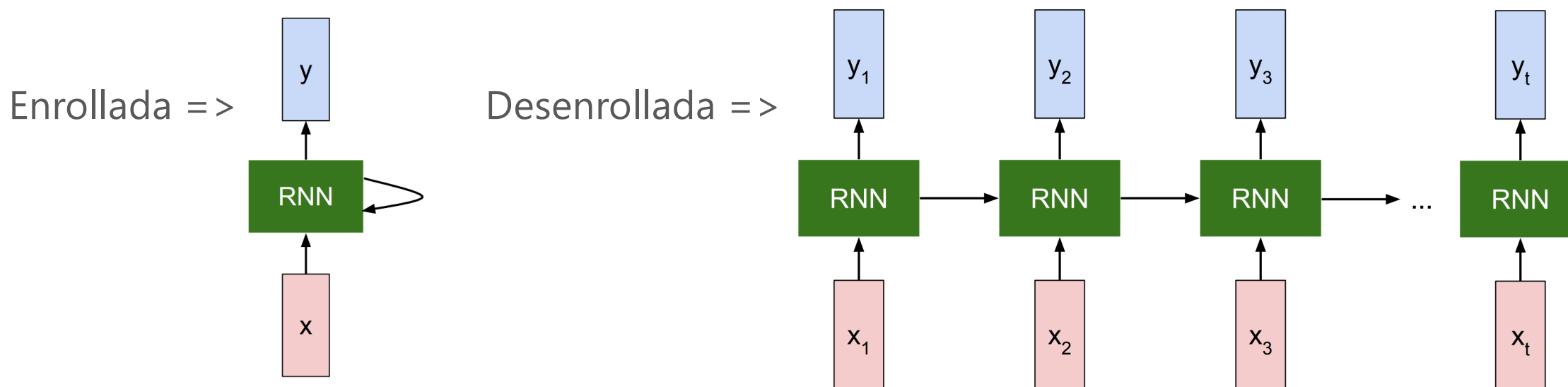


Muchos tipos de celdas:

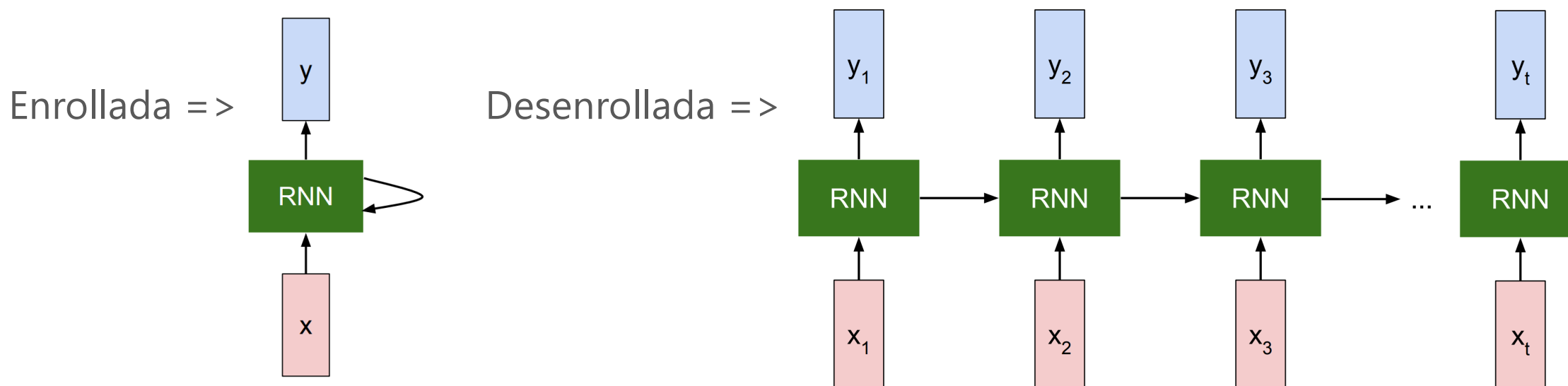
1. Vanilla RNN.
2. LSTM - Memoria a corto y largo plazo.
3. GRU - Unidad recurrente cerrada.



- La RNN es básicamente una caja negra, donde tiene un "estado interno" que se actualiza a medida que se procesa una secuencia. En cada paso de tiempo, alimentamos un vector de entrada en RNN donde modifica ese estado en función de lo que recibe. Cuando ajustamos los pesos de RNN, la RNN mostrará diferentes comportamientos en términos de cómo evoluciona su estado a medida que recibe estas entradas.



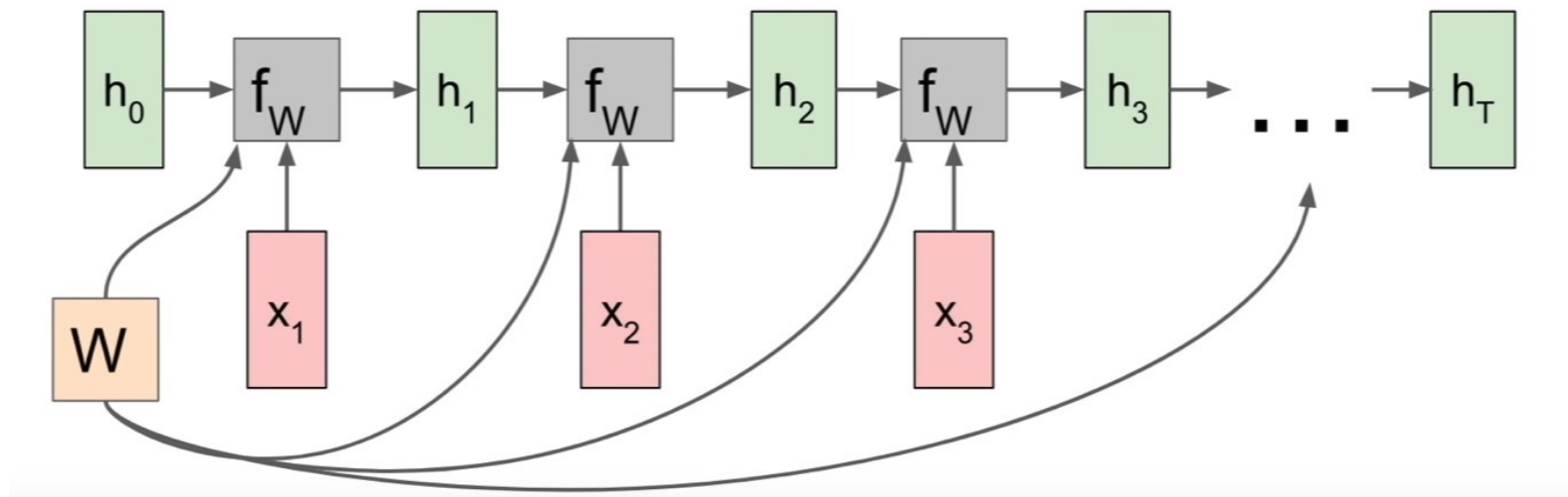
- Si desenrollamos un modelo RNN, entonces hay entradas (por ejemplo, fotograma de video) en diferentes pasos de tiempo que se muestran como  $x_1, x_2, x_3 \dots x_t$ . En la RNN en cada paso de tiempo toma dos entradas, la entrada  $x_i$  y una representación previa de lo que parece hasta ahora (es decir, el historial), para generar una salida  $y_i$  y así actualizar su historial, que se propagará hacia adelante con el tiempo.



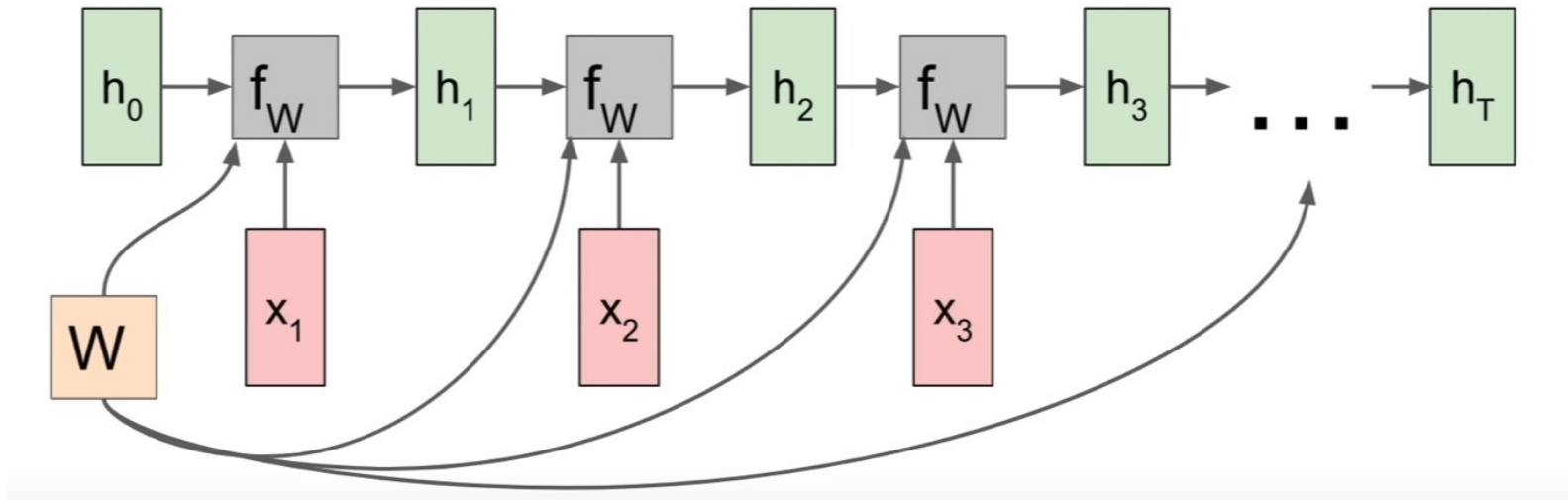
# Formalmente:

- Una RNN se puede representar como una fórmula de recurrencia de alguna función  $f_W$  con parámetros  $W$ :

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



# Formalmente:



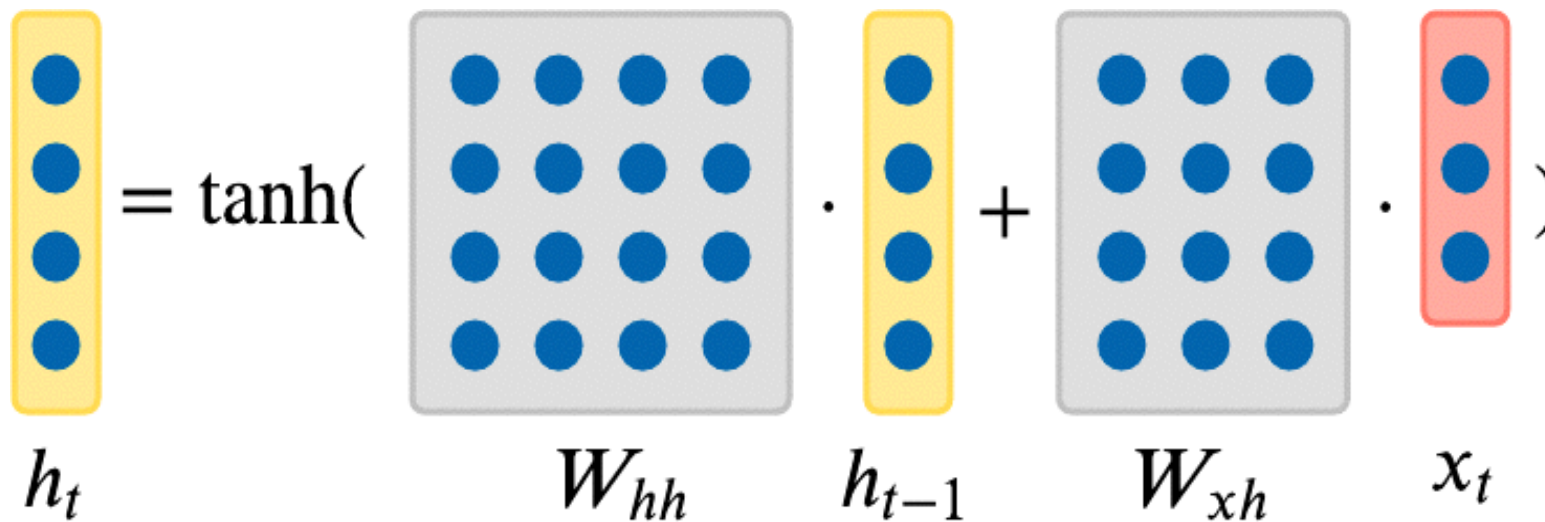
- Donde en cada paso de tiempo recibe algún estado previo como un vector  $h_{t-1}$  de la iteración anterior  $t-1$  y el vector de la entrada actual  $x_t$  para producir el estado actual como un vector  $h_t$ . Se aplica la misma función  $f_W$  con pesos fijos  $W$  en cada paso.

# Vanilla RNN

- En la forma más simple de una RNN, se conoce como Vanilla RNN, la red es solo un único estado oculto  $h$  donde usamos una fórmula de recurrencia que básicamente nos dice cómo debemos actualizar nuestro estado oculto  $h$  en función del estado oculto anterior  $h_{t-1}$  y la entrada actual  $x_t$ .
- En particular, vamos a tener matrices de peso  $W_{hh}$  y  $W_{xh}$  que proyectarán tanto el estado oculto  $h_{t-1}$  del paso de tiempo anterior como la entrada actual  $x_t$  y luego se sumarán y se aplastarán con la función  $\tanh$  para que actualice el estado oculto  $h_t$  en el paso de tiempo  $t$ .
- Se muestra en el siguiente gráfico:

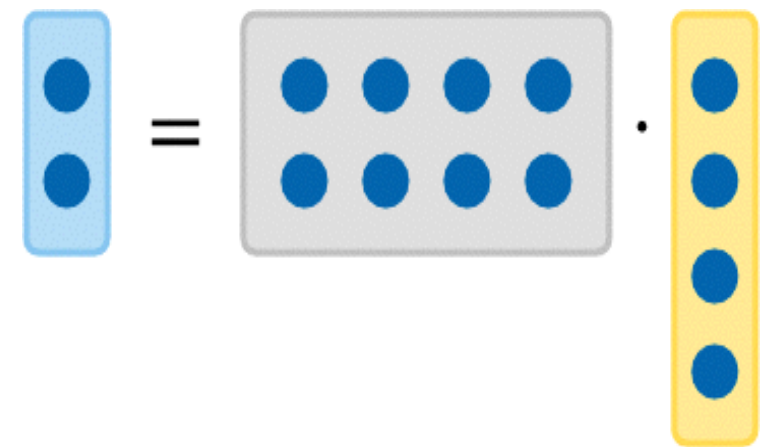
# Vanilla - RNN

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t) \text{ con } h_0 = \vec{0}$$



# Predicción en Vanilla - RNN

- Podemos calcular las predicciones  $y_t$  de  $h_t$  utilizando otra matriz de proyección  $W_{hy}$ .
- Este es el caso completo más simple en el que puede conectar una red neuronal.

$$y_t = W_{hy} h_t$$


$y_t$   $W_{hy}$   $h_t$

# Predicción en Vanilla - RNN

Para una Red Neuronal RNN tipo Vanilla, es decir,  $h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$  y  $h_0 = \vec{0}$  se tiene que:

$$x_1 = \begin{pmatrix} 2 \\ 7 \\ 3 \\ 1 \end{pmatrix}, \quad x_2 = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 1 \end{pmatrix}, \quad W_{xh} = \begin{pmatrix} 2 & 5 & 0 & 3 \\ 0 & 3 & 8 & 2 \\ 3 & 0 & 1 & 1 \\ 0 & 1 & 2 & 0 \\ 1 & 1 & 2 & 1 \end{pmatrix}, \quad W_{hh} = \begin{pmatrix} 1 & 0 & 1 & 1 & 2 \\ 1 & 1 & 4 & 1 & 2 \\ 3 & 0 & 1 & 1 & 1 \\ 1 & 1 & 2 & 0 & 9 \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

1. Calcule  $h_1$  y  $h_2$ .

2. Prediga  $y_2 = W_{hy}h_2$  para  $W_{hy} = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 \end{pmatrix}$ .



## CALCULAR $H_1$ :

$$h_1 = \tanh \left( \begin{pmatrix} 1 & 0 & 1 & 1 & 2 \\ 1 & 1 & 4 & 1 & 2 \\ 3 & 0 & 1 & 1 & 1 \\ 1 & 1 & 2 & 0 & 9 \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix} * \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 2 & 5 & 0 & 3 \\ 0 & 3 & 8 & 2 \\ 3 & 0 & 1 & 1 \\ 0 & 1 & 2 & 0 \\ 1 & 1 & 2 & 1 \end{pmatrix} * \begin{pmatrix} 2 \\ 7 \\ 3 \\ 1 \end{pmatrix} \right)$$

	$W_{hh}$	$h_{t-1} = h_0$	$W_{xh}$	$X_t = X_1$
--	----------	-----------------	----------	-------------

$$h_1 = \begin{pmatrix} 1 \\ 1 \\ 0.9999999958776927 \\ 0.9999999999897818 \\ 0.99999999999999747 \end{pmatrix}$$

## CALCULAR $h_2$ :

$$h_2 = \tanh \left( \begin{pmatrix} 1 & 0 & 1 & 1 & 2 \\ 1 & 1 & 4 & 1 & 2 \\ 3 & 0 & 1 & 1 & 1 \\ 1 & 1 & 2 & 0 & 9 \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix} * \begin{pmatrix} 1 \\ 1 \\ 0.99.. \\ 0.99.. \\ 0.99.. \end{pmatrix} + \begin{pmatrix} 2 & 5 & 0 & 3 \\ 0 & 3 & 8 & 2 \\ 3 & 0 & 1 & 1 \\ 0 & 1 & 2 & 0 \\ 1 & 1 & 2 & 1 \end{pmatrix} * \begin{pmatrix} 0 \\ 1 \\ 0 \\ 1 \end{pmatrix} \right)$$

$W_{hh}$

$h_{t-1} = h_1$

$W_{xh}$

$X_t = X_2$

$$h_2 = \begin{pmatrix} 0.9999999999897 \\ 0.9999999999986 \\ 0.9999983269356217 \\ 0.9999999999985864 \\ 0.9999983269356217 \end{pmatrix}$$

Predecir  $y_2 = W_{hy} * h_2$ , usando como base la siguiente matriz para  $W_{hy}$ :

$$W_{hy} = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 \end{pmatrix}$$

Procedemos a calcular la multiplicación de  $(W_{hy} * h_2)$  de la siguiente forma:

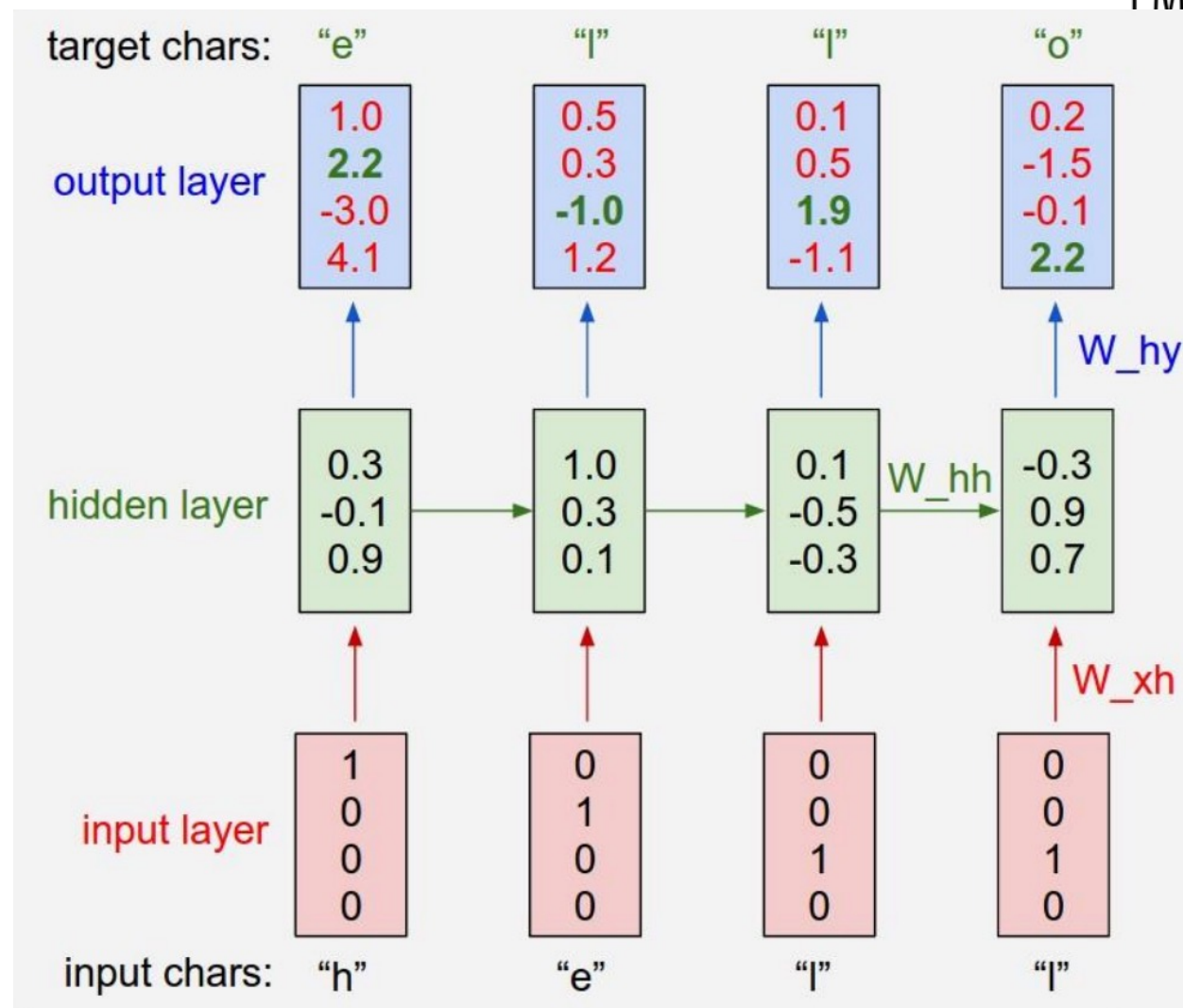
$$y_2 = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 \end{pmatrix} * \begin{pmatrix} 0.999999999999897 \\ 0.999999999999986 \\ 0.9999983269356217 \\ 0.999999999999985864 \\ 0.9999983269356217 \end{pmatrix}$$

$$y_2 = \begin{pmatrix} 2.997 \\ 2.997 \end{pmatrix}$$

# Ejemplo: Una RNN para predecir la siguiente letra de una secuencia de letras

- Supongamos que tenemos una secuencia de letras de entrenamiento de una sola palabra "hello" y tenemos un conjunto de letras  $V \in \{"h","e","l","o"\}$  de 4 caracteres que es todo el conjunto de datos.
- Vamos a intentar entrenar una RNN para aprender a predecir la siguiente letra de la secuencia en estos datos de entrenamiento.

# Una RNN para predecir la siguiente letra:



## Ejemplo: Una RNN para predecir la siguiente letra de una secuencia de letras

- La RNN se alimenta con una letra a la vez, primero la "h", luego la "e", luego la "l" y finalmente la "l" (queremos predecir que la siguiente es una "o"). Todos los caracteres están codificados en la representación de lo que se llama un "one-hot vector", donde solo se activa un bit único del vector para cada letra en el vocabulario:

$$h = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad e = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \quad l = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \quad o = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

## Ejemplo: Una RNN para predecir la siguiente letra de una secuencia de letras

- Vamos a usar la fórmula de recurrencia de la RNN. Supongamos que comenzamos con  $h$  como un vector de tamaño 3 con todas entradas igual a cero.
- Al usar esta fórmula de recurrencia fija, vamos a terminar con una representación en 3 dimensiones del siguiente estado oculto  $h$  que básicamente en cualquier momento resume todas las letras que han llegado hasta entonces, veamos:

## Ejemplo: Una RNN para predecir la siguiente letra de una secuencia

Donde  $W_{hh}$  y  $W_{xh}$  son matrices fijas de pesos, de tamaño  $4 \times 3$ , que fueron calculados en el proceso de entrenamiento de la Red Neuronal (aquí no se presentan).

$$\begin{bmatrix} 0.3 \\ -0.1 \\ 0.9 \end{bmatrix} = f_W \left( W_{hh} \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} + W_{xh} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \right)$$

$$\begin{bmatrix} 1.0 \\ 0.3 \\ 0.1 \end{bmatrix} = f_W \left( W_{hh} \begin{bmatrix} 0.3 \\ -0.1 \\ 0.9 \end{bmatrix} + W_{xh} \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \right)$$

$$\begin{bmatrix} 0.1 \\ -0.5 \\ -0.3 \end{bmatrix} = f_W \left( W_{hh} \begin{bmatrix} 1.0 \\ 0.3 \\ 0.1 \end{bmatrix} + W_{xh} \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \right)$$

$$\begin{bmatrix} -0.3 \\ 0.9 \\ 0.7 \end{bmatrix} = f_W \left( W_{hh} \begin{bmatrix} 0.1 \\ -0.5 \\ -0.3 \end{bmatrix} + W_{xh} \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \right)$$



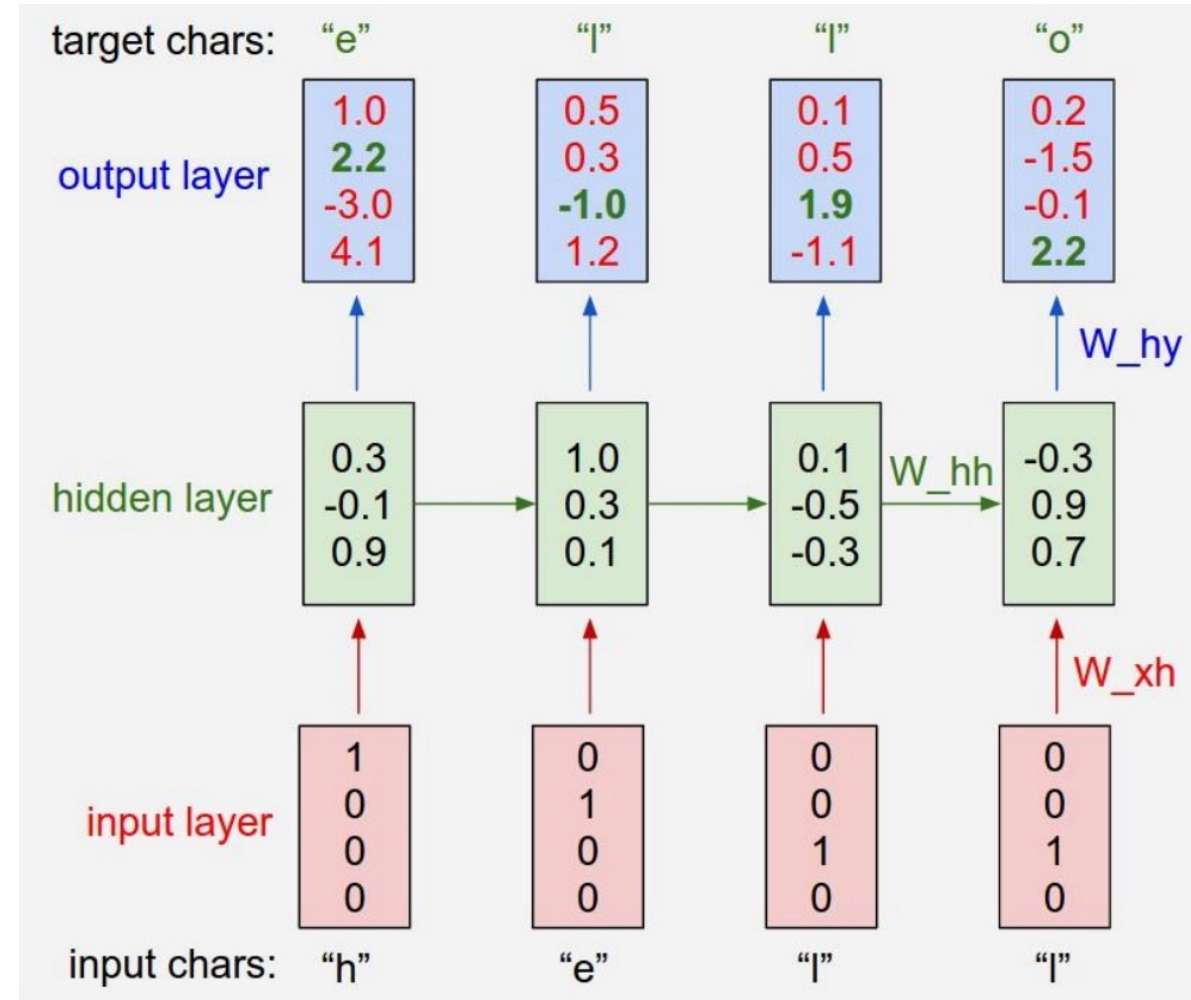


$$\begin{bmatrix} 0.3 \\ -0.1 \\ 0.9 \end{bmatrix} = f_W \left( W_{hh} \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} + W_{xh} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \right)$$

$$\begin{bmatrix} 1.0 \\ 0.3 \\ 0.1 \end{bmatrix} = f_W \left( W_{hh} \begin{bmatrix} 0.3 \\ -0.1 \\ 0.9 \end{bmatrix} + W_{xh} \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \right)$$

$$\begin{bmatrix} 0.1 \\ -0.5 \\ -0.3 \end{bmatrix} = f_W \left( W_{hh} \begin{bmatrix} 1.0 \\ 0.3 \\ 0.1 \end{bmatrix} + W_{xh} \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \right)$$


$$\begin{bmatrix} -0.3 \\ 0.9 \\ 0.7 \end{bmatrix} = f_W \left( W_{hh} \begin{bmatrix} 0.1 \\ -0.5 \\ -0.3 \end{bmatrix} + W_{xh} \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \right)$$



Donde  $W_{hh}$ ,  $W_{xh}$  y  $W_{hy}$  son matrices fijas de pesos de tamaño 4 x 3 cuyos valores se calcularon en el proceso de entrenamiento de la Red Neuronal.

## Ejemplo: Una RNN para predecir la siguiente letra de una secuencia de letras

- A medida que se aplica la recurrencia de la RNN en cada paso de tiempo, vamos a predecir cuál debería ser la siguiente letra de la secuencia en cada paso de tiempo.
- Dado que tenemos cuatro letras en el vocabulario  $V$ , vamos a predecir el vector 4-dimensional binario en cada paso de tiempo.
- *En el primer paso del tiempo alimentamos en con una letra  $h$ . La RNN con su configuración actual de pesos calculó el siguiente vector salida, observe que el **máximo** es 4.1 y está en la cuarta entrada del vector:*

$$\begin{bmatrix} 1.0 \\ 2.2 \\ -3.0 \\ 4.1 \end{bmatrix}$$


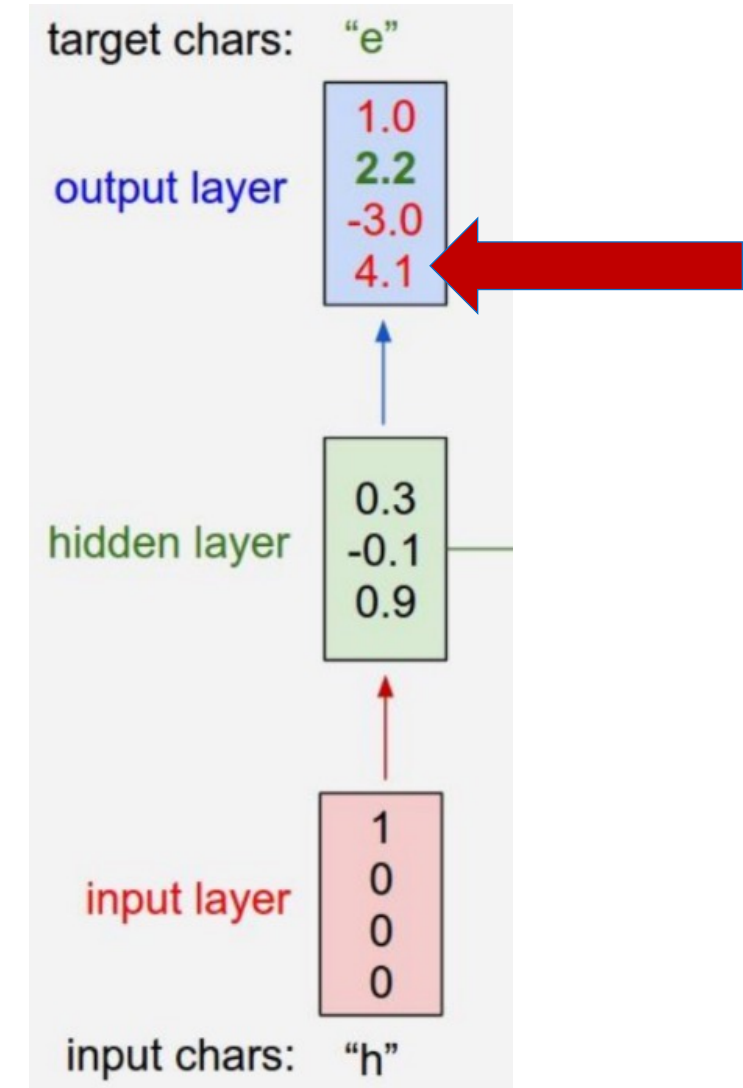
## Ejemplo: Una RNN para predecir la siguiente letra de una secuencia

- Esto significa que la RNN está prediciendo erróneamente la segunda letra de la secuencia como una letra "o", pero debe ser una letra "e" por lo que se toma como salida el 2.2 marcado en verde.

$$\begin{bmatrix} 1.0 \\ 2.2 \\ -3.0 \\ 4.1 \end{bmatrix}$$

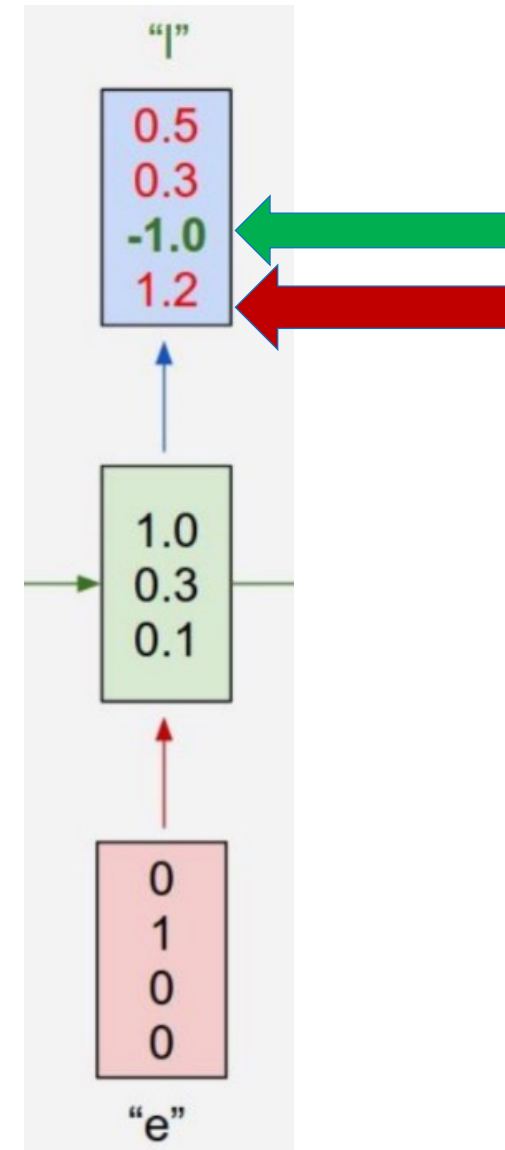
← Green arrow pointing to 2.2

← Red arrow pointing to 4.1



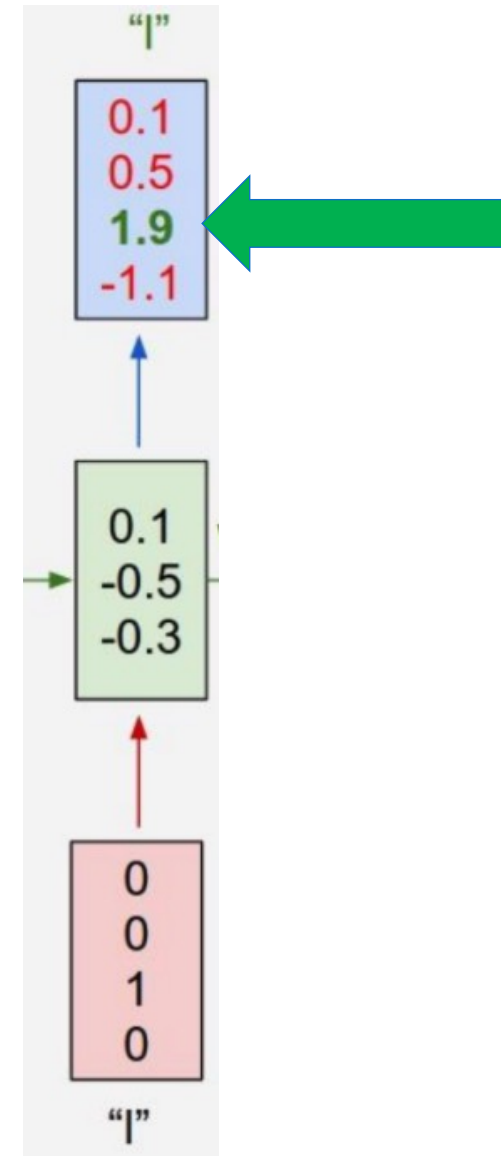
## Ejemplo: Una RNN para predecir la siguiente letra de una secuencia

- De igual manera, la RNN está prediciendo erróneamente la tercera letra de la secuencia como una letra "o", pero debe ser una letra "l", por lo que se toma como salida el -1.0 marcado en verde.



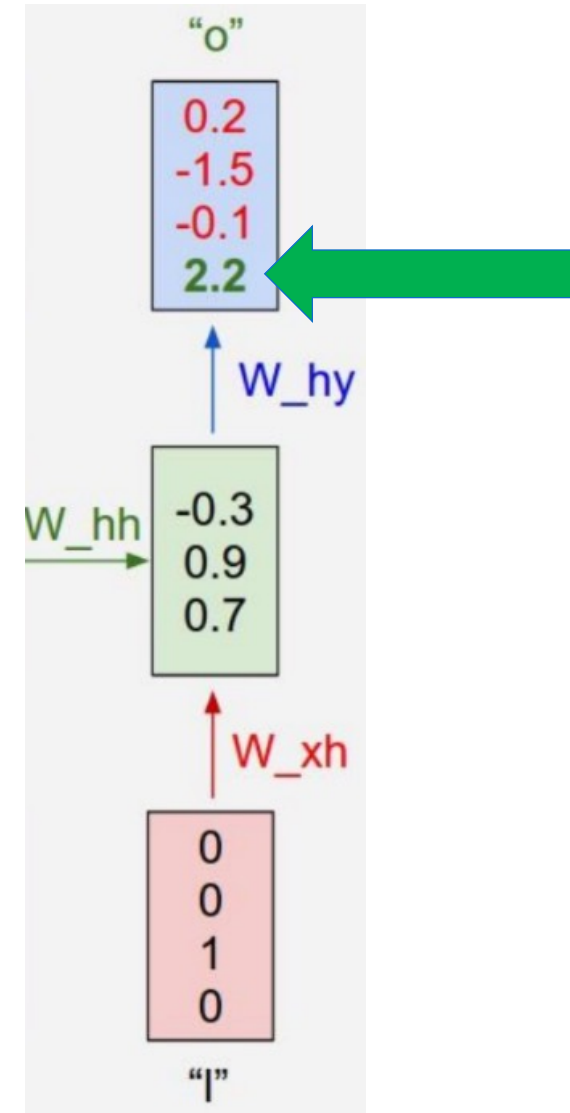
## Ejemplo: Una RNN para predecir la siguiente letra de una secuencia

- Para la cuarta letra la predicción sí es correcta, la RNN está prediciendo como cuarta letra de la secuencia una letra "l", por lo que se toma como salida el 1.9 marcado en verde.



## Ejemplo: Una RNN para predecir la siguiente letra de una secuencia

- Para la quinta letra la predicción también es correcta, la RNN está prediciendo como quita letra de la secuencia una letra "o", por lo que se toma como salida el 2.2 marcado en verde.



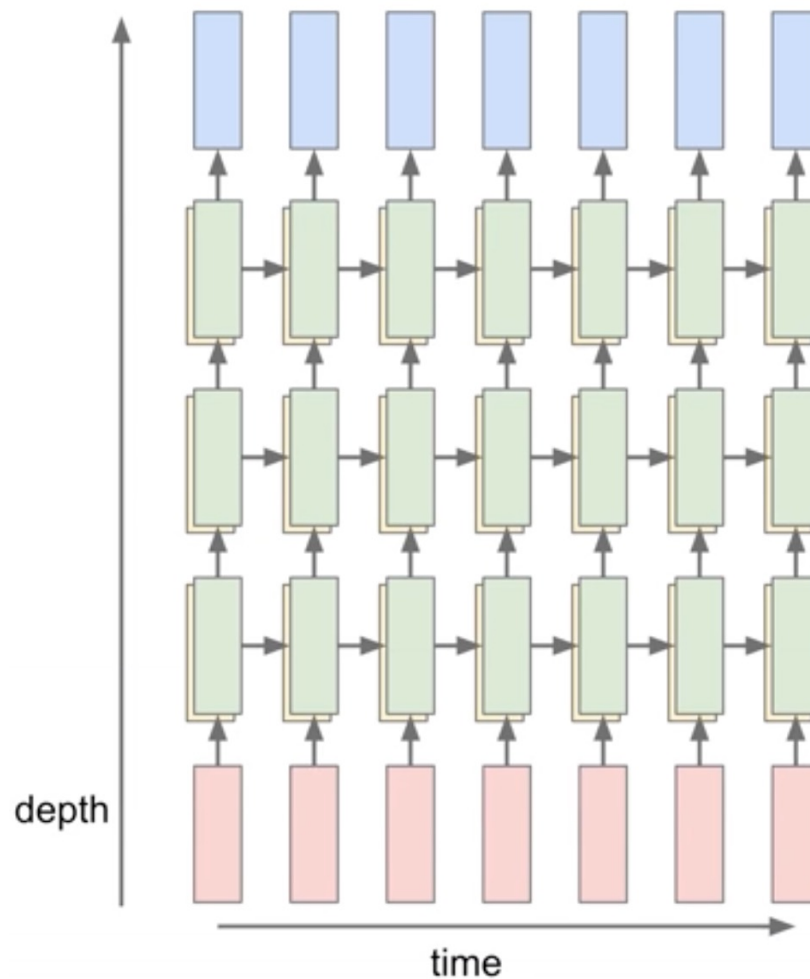
# Vanilla con Sesgo (bias)

- Las CNN se caracterizan por la **convolución** de los núcleos a través del mapa de características de entrada, por otra parte la salida de la RNN es una función **no solo de la entrada actual, sino también de la salida anterior o del estado oculto**.
- Dado que la salida anterior también es una función de la entrada anterior, la salida actual también es una función de la salida y la entrada anteriores y así sucesivamente. La capa SimpleRNN en Keras es una versión simplificada del verdadero RNN. La siguiente ecuación describe la **salida** de SimpleRNN:

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

- En esta ecuación,  $b$  es el sesgo (bias), mientras que  $W$  y  $U$  se denominan **kernel recurrente** (pesos para la salida anterior) y **kernel** (pesos para la entrada actual), respectivamente. El subíndice  $t$  se utiliza para indicar la posición en la secuencia.

# RNN profundas (RNN Multicapa)





# Entrenamiento y Optimización

- Como en las redes neuronales que ya hemos visto antes, las RNN también actualizan sus parámetros mediante **retropropagación** al encontrar el **gradiente del error (pérdida) con respecto a los pesos**.
- Aquí, sin embargo, se lo conoce como **Retropropagación a través del tiempo (BPTT)** porque cada nodo en el RNN tiene un paso de tiempo.
- En este caso, usando BPTT, se quiere averiguar cuánto afectan las unidades ocultas y la salida al error total, así como cuánto afecta el cambio de pesos  $W_{hh}$  y  $W_{xh}$  a la salida. Como los  $W$ s son constantes en toda la red debemos retroceder hasta el paso de tiempo inicial para actualizarlo.

# Entrenamiento y optimización

Al retropropagar en RNN, se aplica nuevamente la regla de la cadena. Lo que dificulta el entrenamiento de los RNN es que la función de pérdida depende no solo de la activación de la capa de salida, sino también de la activación de la capa oculta actual y su efecto en la capa oculta en el siguiente paso de tiempo. Más adelante veremos en detalle, en la siguiente ecuación, cómo se calcula.

$$\frac{\partial E}{\partial W} = \sum_{i=1}^{\tau} \frac{\partial E_i}{\partial W}$$

# Red de Memoria de Corto y Largo Plazo (Long-Term Short Memory, LTSM)

- Hasta ahora solo hemos visto una fórmula de recurrencia simple para las RNN tipo Vanilla. En la práctica, en realidad rara vez se usan la fórmula RNN tipo Vanilla. En su lugar, se usan las RNN de memoria a largo y corto plazo (LSTM).
- Las LTSM surgen para resolver el problema de la desaparición del gradiente en las RNN tipo Vanilla (RNN Vanishing Gradient Problem).

# Desaparición del gradiente en las RNN tipo Vanilla (RNN Vanishing Gradient Problem)

- Un bloque de una RNN toma la entrada  $x_t$  y la representación oculta anterior  $h_{t-1}$  así aprende y luego se pasa a través de  $\tanh$  para producir la representación oculta  $h_t$  como se muestra en la ecuación a continuación:

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

## Desaparición del gradiente en las RNN tipo Vanilla (RNN Vanishing Gradient Problem)

- Para la propagación hacia atrás, examinemos cómo la salida en el último paso de tiempo afecta los pesos en el primer paso de tiempo.
- La derivada parcial de  $h_t$  con respecto a  $h_{t-1}$  se escribe como:

$$\frac{\partial h_t}{\partial h_{t-1}} = \tanh' (W_{hh}h_{t-1} + W_{xh}x_t) W_{hh}$$

# Desaparición del gradiente en las RNN tipo Vanilla (RNN Vanishing Gradient Problem)

- Actualizamos los pesos  $W_{hh}$  calculando la derivada de la función de pérdida o de Error (denotada por  $E_t$ ) en el último paso de tiempo con respecto a  $W_{hh}$  como sigue:

$$\begin{aligned}\frac{\partial E_t}{\partial W_{hh}} &= \frac{\partial E_t}{\partial h_t} \frac{\partial h_t}{\partial h_{t-1}} \dots \frac{\partial h_1}{\partial W_{hh}} \\ &= \frac{\partial E_t}{\partial h_t} \left( \prod_{t=2}^T \frac{\partial h_t}{\partial h_{t-1}} \right) \frac{\partial h_1}{\partial W_{hh}} \\ &= \frac{\partial E_t}{\partial h_t} \left( \prod_{t=2}^T \tanh' (W_{hh} h_{t-1} + W_{xh} x_t) W_{hh}^{T-1} \right) \frac{\partial h_1}{\partial W_{hh}}\end{aligned}$$

# Desaparición del gradiente en las RNN tipo Vanilla (RNN Vanishing Gradient Problem)

- **Vanishing Gradient Problem:** Vemos que  $\tanh'(W_{hh}h_{t-1} + W_{xh}x_t)$  casi siempre será menor que 1 porque  $\tanh$  siempre está entre uno negativo y uno.
- Por lo tanto, a medida que  $t$  se hace más grande (es decir, pasos de tiempo más largos), el gradiente  $(\partial E_t / \partial W_{hh})$  se reducirá en valor y se acercará a cero.
- Esto conducirá a un problema de la desaparición del gradiente. Esto es problemático cuando modelamos secuencias largas porque las actualizaciones serán extremadamente lentas.

# Eliminación de la no linealidad de tanh

- Eliminamos la no linealidad (*tanh*) para resolver el problema de desaparición de gradiente, entonces nos quedaremos con:

$$\frac{\partial E_t}{\partial W_{hh}} \cong \frac{\partial E_t}{\partial h_t} \left( \prod_{t=2}^T W_{hh}^{T-1} \right) \frac{\partial h_1}{\partial W_{hh}}$$

- **Gradientes explosivos:** Si el valor propio más grande de  $W_{hh}$  es mayor que 1, entonces los gradientes explotarán y el modelo obtendrá gradientes muy grandes que a menudo conducen a obtener gradientes que son NaNs.
- **Gradientes muy pequeños:** Si el valor propio más grande de  $W_{hh}$  es menor que 1, entonces tendremos un problema de desaparición del gradiente lo que ralentizará significativamente el aprendizaje.



# Formulación de las Redes LSTM

- Sin embargo, dado que el problema de desaparición del gradiente en los casos en que el mayor valor propio de la matriz  $W_{hh}$  es menor que uno, las redes LSTM fueron diseñadas para evitar este problema.
- En el paso  $t$ , hay un estado oculto  $h_t$  y una celda de estado  $c_t$ . Tanto  $h_t$  como  $c_t$  son vectores de tamaño  $n$ . Una distinción de LSTM de Vanilla RNN es que LSTM tiene esta celda estado adicional  $c_t$  que intuitivamente se puede pensar que  $c_t$  almacena información (memoria) a largo plazo.
- LSTM puede leer, borrar y escribir información a través de esta celda  $c_t$ . La forma en que LSTM altera la celda  $c_t$  es a través de tres puertas (vectores) especiales:  $i$ ,  $f$ ,  $o$  que corresponden a las puertas de "*entrada*" (input), "*olvido*" (forget) y "*salida*" (output). Los valores de estas puertas varían de cerradas (0) a abiertas (1). Todas las puertas  $i$ ,  $f$ ,  $o$  son vectores de tamaño  $n$ .

# Formulación de las Redes LTSM

- En cada paso de tiempo tenemos un vector de entrada  $x_t$  el estado oculto anterior  $h_{t-1}$  y el estado de celda anterior  $c_{t-1}$  entonces LSTM calcula el siguiente estado oculto  $h_t$  así el siguiente estado de celda  $c_t$  en el paso  $t$  se calcula la siguiente manera:

$$f_t = \sigma (W_{hf}h_{t-1} + W_{xf}x_t)$$

$$i_t = \sigma (W_{hi}h_{t-1} + W_{xi}x_t)$$

$$o_t = \sigma (W_{ho}h_{t-1} + W_{xo}x_t)$$

$$g_t = \tanh (W_{hg}h_{t-1} + W_{xg}x_t)$$

$$h_0 = \vec{0}$$

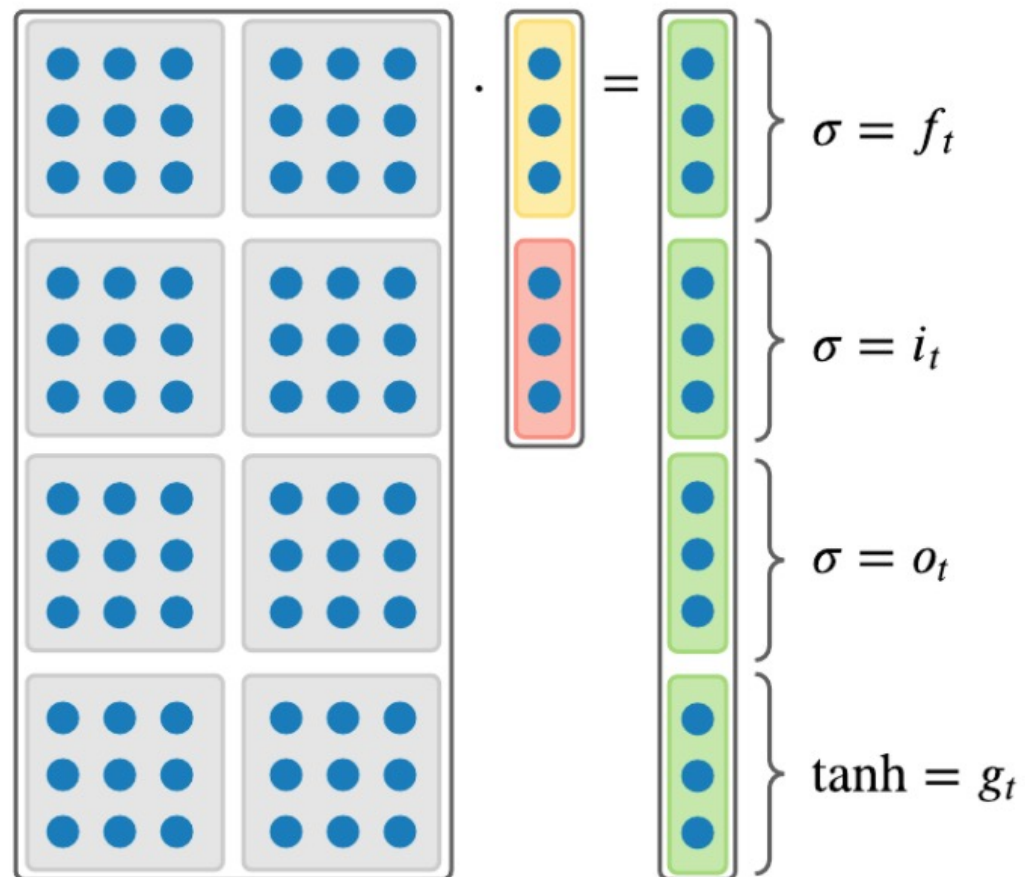
$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

$$c_t = f_t \odot c_{t-1} + i_t \odot g_t$$

$$h_t = o_t \odot \tanh (c_t)$$

$$c_0 = \vec{0}$$

# Formulación de las Redes LSTM

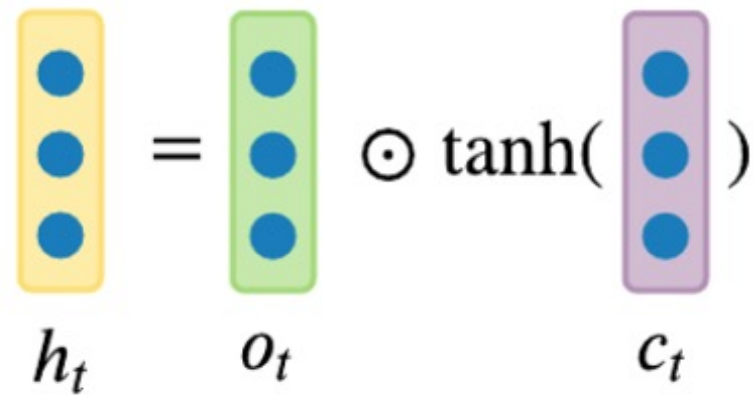
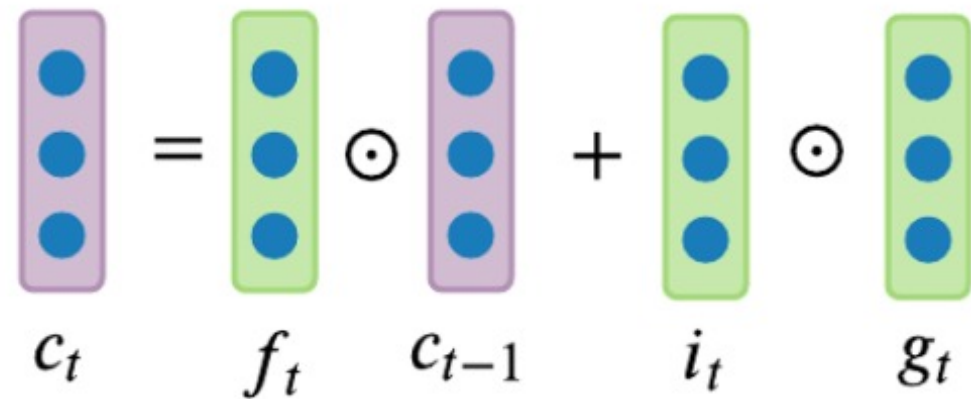


$$\begin{matrix} W_{hf} & W_{xf} \\ W_{hi} & W_{xi} \\ W_{ho} & W_{xo} \\ W_{hg} & W_{xg} \end{matrix} \quad \begin{matrix} h_{t-1} \\ x_t \end{matrix}$$

## Formulación de las Redes LSTM

$$c_t = f_t \odot c_{t-1} + i_t \odot g_t$$

$$h_t = o_t \odot \tanh(c_t)$$



# Formulación de las Redes LSTM

$$c_t = f_t \odot c_{t-1} + i_t \odot g_t$$

$$h_t = o_t \odot \tanh(c_t)$$

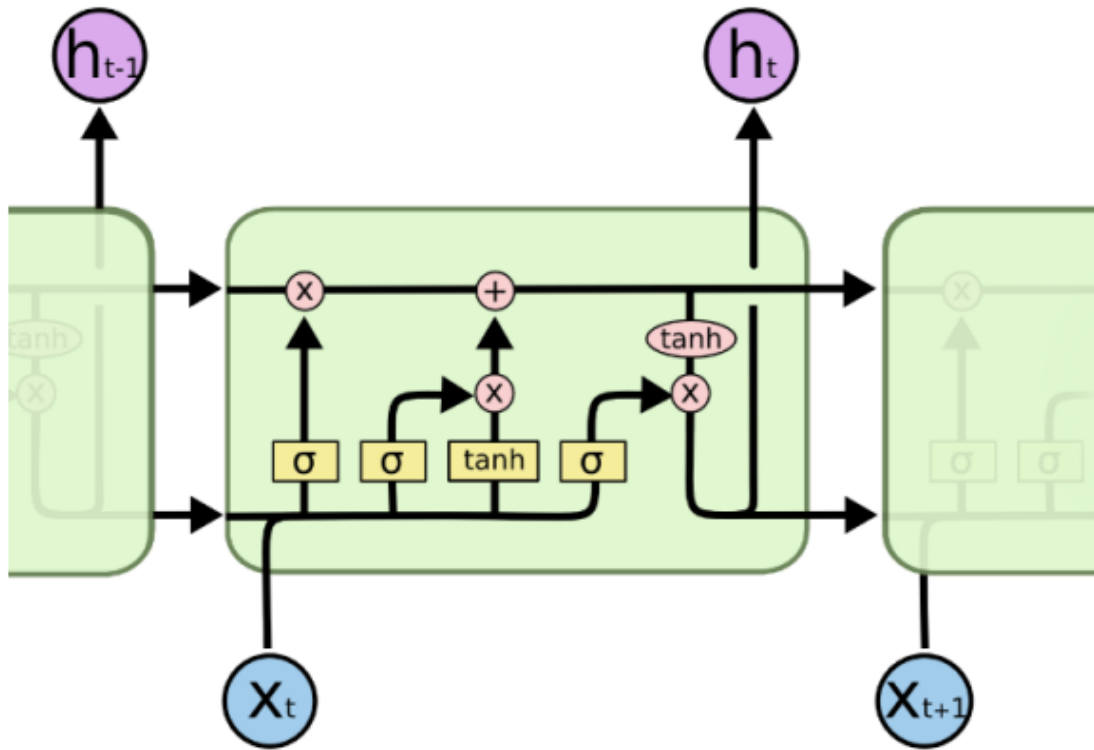
- Donde  $\odot$  es un producto Hadamard.  $g_t$  es una especie caché de cálculo intermedio que luego se usa en el siguiente producto vectorial con  $o_t$ .
- Donde, por ejemplo, el producto de Hadamard para una matriz  $A$  de  $3 \times 3$  con una matriz  $B$  de  $3 \times 3$  es:

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \odot \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix} = \begin{bmatrix} a_{11} \cdot b_{11} & a_{12} \cdot b_{12} & a_{13} \cdot b_{13} \\ a_{21} \cdot b_{21} & a_{22} \cdot b_{22} & a_{23} \cdot b_{23} \\ a_{31} \cdot b_{31} & a_{32} \cdot b_{32} & a_{33} \cdot b_{33} \end{bmatrix}$$

# Formulación de las Redes LTSM

- Todos los valores del vector de las compuertas  $f$ ,  $i$ ,  $o$  varían de 0 a 1, esto debido a que fueron aplastados por la función sigmoide  $\sigma$ , cuando se multiplican, entonces se tiene que:
- **Puerta de Olvido** (Forget Gate):  $f_t$  en el tiempo o paso  $t$  controla cuánta información debe "eliminarse" del estado de celda anterior  $c_{t-1}$ .
- **La Puerta de Entrada** (Input Gate): en el paso  $t$  controla cuánta información debe "agregarse" al siguiente estado o celda  $c_t$  desde el estado oculto anterior  $h_{t-1}$  y la entrada  $x_t$ .
- **La Puerta de Salida** (Output Gate):  $o_t$  en el paso  $t$  controla cuánta información debe "mostrarse" como salida en el estado oculto actual  $h_t$ .

# En Resumen: Redes LSTM



$$f_t = \sigma (W_{hf}h_{t-1} + W_{xf}x_t)$$

$$i_t = \sigma (W_{hi}h_{t-1} + W_{xi}x_t)$$

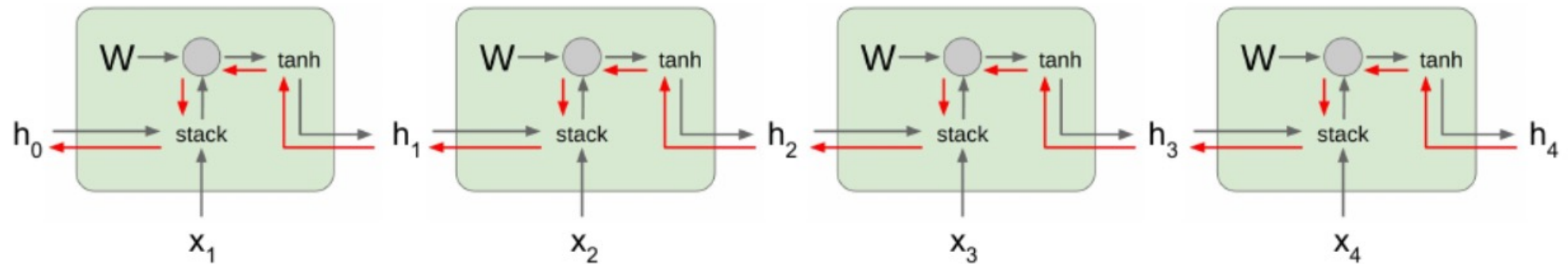
$$o_t = \sigma (W_{ho}h_{t-1} + W_{xo}x_t)$$

$$g_t = \tanh (W_{hg}h_{t-1} + W_{xg}x_t)$$

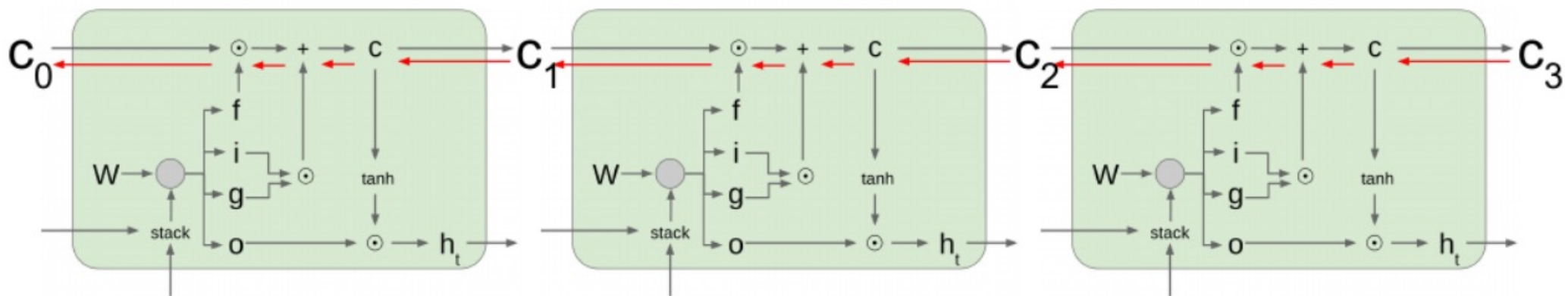
$$c_t = f_t \odot c_{t-1} + i_t \odot g_t$$

$$h_t = o_t \odot \tanh (c_t)$$

# Redes Vanilla



# Redes LSTM





# Gracias ....

# GRACIAS....