

# Redes Recurrentes

Orlando Ramos Flores

# Contenido

- Introducción
- Despliegue de grafos
- Redes Neuronales Recurrentes
  - Arquitectura general
  - Propagación hacia adelante
  - Retropropagación a través del tiempo
  - Otras arquitecturas
  - Funciones de Activación
  - Ventajas y desventajas

# Introducción

# Introducción

La Redes Neuronales Recurrentes (RNN), son una familia de redes neuronales para el procesamiento de datos secuenciales.

## Examples of sequence data

Speech recognition



Music generation

Sentiment classification

"There is nothing to like  
in this movie."

DNA sequence analysis

AGCCCCTGTGAGGAACTAG

Machine translation

Voulez-vous chanter avec  
moi?

Video activity recognition



Name entity recognition

Yesterday, Harry Potter  
met Hermione Granger.

y  
"The quick brown fox jumped  
over the lazy dog."



AGCCCCTGTGAGGAACTAG

Do you want to sing with  
me?

Running

Yesterday, **Harry Potter**  
met **Hermione Granger**.  
Andrew Ng

# Introducción

- Una Red Neuronal Recurrente (RNN) es una red neuronal especializada para procesar una secuencia de valores  $x^{(l)}, \dots, x^{(\tau)}$ .
- Para pasar de redes multicapa a redes recurrentes, debemos compartir parámetros en diferentes partes del modelo.
- Compartir parámetros hace posible extender y aplicar el modelo a ejemplos de diferentes formas (diferentes longitudes) y generalizar a través de ellos.

# Introducción

- Si tuviéramos parámetros separados para cada valor del índice de tiempo, no podríamos generalizar a longitudes de secuencia no vistas durante el entrenamiento, ni compartir la fuerza estadística a través de diferentes longitudes de secuencia y en diferentes posiciones en el tiempo.
- Tal intercambio es particularmente importante cuando una información específica puede ocurrir en múltiples posiciones dentro de la secuencia.
- Por ejemplo: *Fui a Nepal en el 2009*, y *En el 2009 fui a Nepal*. ¿En qué año fui a Nepal?, en el 2009, no importa si se encuentra en la sexta posición en la oración o en la tercera.

# Introducción

- Una red FeedForward tradicional totalmente conectada (FC) tendría parámetros separados para cada función de entrada, por lo que tendría que aprender todas las reglas del idioma por separado en cada posición de la oración.
- En comparación, una red neuronal recurrente comparte los mismos pesos en varios pasos de tiempo.
- Las redes recurrentes comparten parámetros de una manera diferente. Cada miembro de la salida es una función de los miembros anteriores de la salida.
- Cada miembro de la salida se produce usando la misma regla de actualización aplicada a las salidas anteriores.
- Esta formulación recurrente da como resultado el intercambio de parámetros a través de un grafo computacional muy profundo.

# Despliegue de grafos



# Despliegue (unfolding) de grafos

- Un grafo es una forma de formalizar la estructura de un conjunto de cálculos, como los involucrados en el mapeo de entradas y parámetros a salidas y pérdidas.
- La idea de desarrollar un cálculo recursivo o recurrente en un grafo que tiene una estructura repetitiva, normalmente corresponde a una cadena de eventos.
- El despliegue de este grafo da como resultado el intercambio de parámetros a través de una estructura de red profunda.

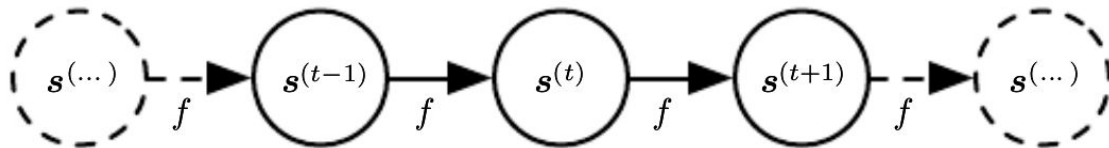
# Despliegue de grafos

- Considerar la forma clásica de un sistema dinámico:  $s^{(t)} = f(s^{(t-1)}; \theta)$
- Donde  $s^{(t)}$  es llamado el estado del sistema.
- La ecuación es recurrente, porque la definición de  $s$  en el tiempo  $t$  se refiere a la misma definición en el tiempo  $t-1$ .
- Para un número finito de pasos de tiempo  $T$ , el grafo se puede desplegar aplicando la definición  $T-1$  veces.
- Por ejemplo, para  $T=3$  obtenemos:

$$\begin{aligned} s^{(3)} &= f(s^{(2)}; \theta) \\ &= f(f(s^{(1)}; \theta); \theta) \end{aligned}$$

# Despliegue de grafos

- Desplegar la ecuación aplicando repetidamente la definición de esta manera ha producido una expresión que no implica recurrencia.
- Tal expresión ahora se puede representar mediante un grafo acíclico dirigido tradicional.



- Cada nodo representa el estado en algún momento  $t$  y la función  $f$  asigna el estado en  $t$  al estado en  $t + 1$ .
- Los mismos parámetros (el mismo valor de  $\theta$  usado para parametrizar  $f$ ) se usan para todos los pasos de tiempo.

# Despliegue de grafos

- Consideremos un sistema dinámico impulsado por una señal externa  $x^{(t)}$ :

$$s^{(t)} = f(s^{(t-1)}, x^{(t)}; \theta)$$

- Aquí, vemos que el estado ahora contiene información sobre toda la secuencia pasada.
- Así como casi cualquier función puede considerarse una red feedforward, esencialmente cualquier función que involucre recurrencia puede considerarse una red neuronal recurrente.

# Despliegue de grafos

- Para indicar que el estado son las unidades ocultas de la red, ahora reescribimos la ecuación anterior usando la variable  $\mathbf{h}$  para representar el estado.
- Muchas redes neuronales recurrentes utilizan la siguiente ecuación o una ecuación similar para definir los valores de sus unidades ocultas.

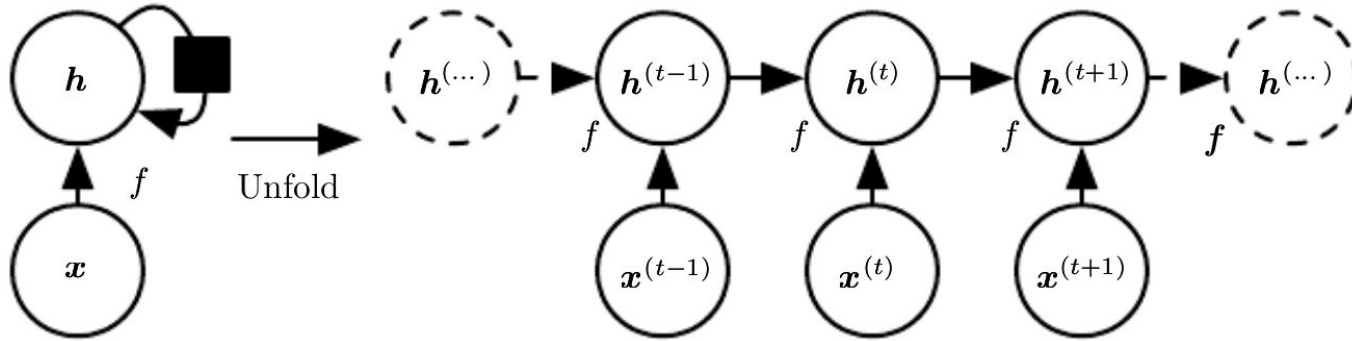
$$\mathbf{h}^{(t)} = f(\mathbf{h}^{(t-1)}, \mathbf{x}^{(t)}; \boldsymbol{\theta})$$

# Despliegue de grafos

- Cuando la Red Neuronal Recurrente se entrena para realizar una tarea que requiere predecir el futuro a partir del pasado, la red normalmente aprende a usar  $\mathbf{h}^{(t)}$  como una especie de resumen con pérdida de los aspectos relevantes de la tarea de la secuencia de entradas pasadas hasta  $t$ .
- Este resumen es necesariamente con pérdida, ya que mapea una secuencia de longitud arbitraria  $(\mathbf{x}^{(t)}, \mathbf{x}^{(t-1)}, \mathbf{x}^{(t-2)}, \dots, \mathbf{x}^{(2)}, \mathbf{x}^{(1)})$  a un vector de longitud fija  $\mathbf{h}^{(t)}$ .

# Despliegue de grafos

- Una RNN sin salidas

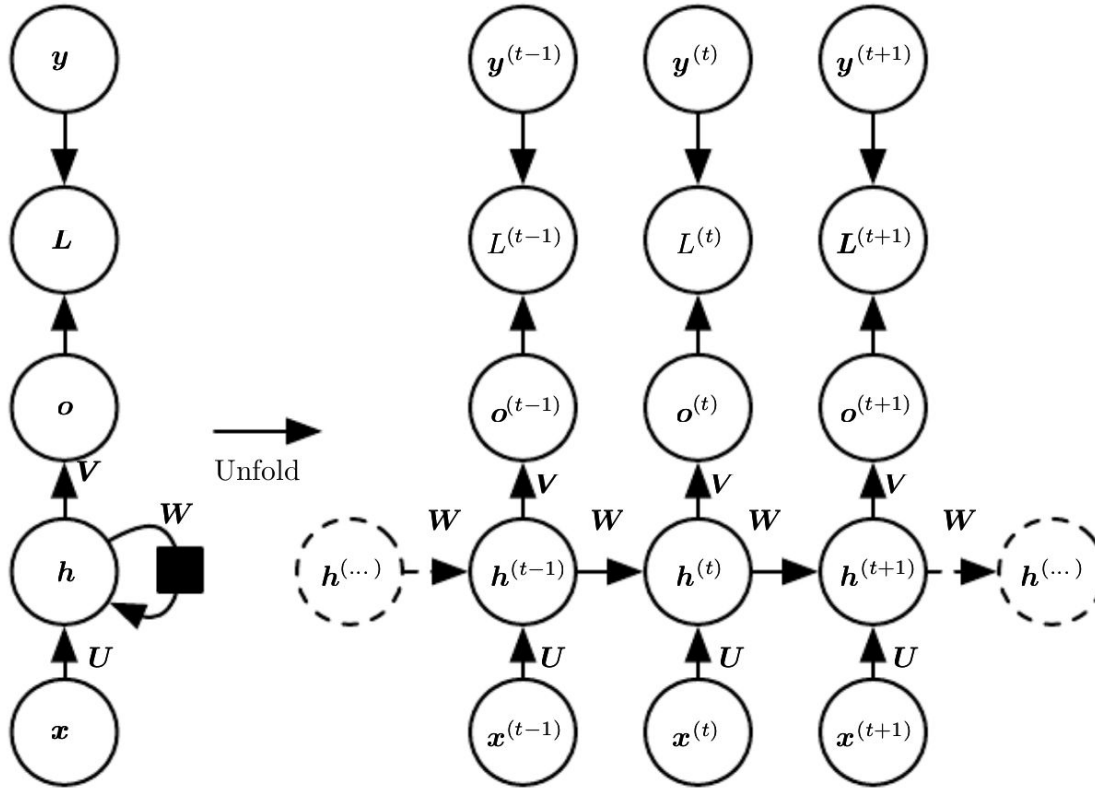


- El cuadrado negro del diagrama es para indicar que se lleva a cabo una iteración con un retraso de un solo paso de tiempo, desde el estado en el tiempo  $t$  hasta el estado en el tiempo  $t + 1$ .

# Redes Neuronales Recurrentes



# Redes Recurrentes: Arquitectura General



$x^t$ : entradas

$h^t$ : capas ocultas de activación

$o^t$ : salidas

$y^t$ : los objetivos (targets)

$L^t$ : pérdidas (internamente calcula  $\hat{y} = \text{softmax}(o)$  y se hace la comparación con  $y$ )

$U$ : matriz de pesos (input to hidden)

$W$ : matriz de pesos (hidden to hidden)

$V$ : matriz de pesos (hidden to output)

Redes recurrentes que producen una salida en cada paso de tiempo y tienen conexiones recurrentes entre unidades ocultas

# Redes Recurrentes: Forward

Para cada paso de tiempo de  $t=1$  a  $t=T$ , aplicamos las siguientes ecuaciones:

$$\begin{aligned} \mathbf{a}^{(t)} &= \mathbf{b} + \mathbf{W}\mathbf{h}^{(t-1)} + \mathbf{U}\mathbf{x}^{(t)} \\ \mathbf{h}^{(t)} &= \tanh(\mathbf{a}^{(t)}) \\ \mathbf{o}^{(t)} &= \mathbf{c} + \mathbf{V}\mathbf{h}^{(t)} \\ \hat{\mathbf{y}}^{(t)} &= \text{softmax}(\mathbf{o}^{(t)}) \end{aligned}$$

donde los parámetros son los vectores de *bias*  $\mathbf{b}$  y  $\mathbf{c}$  junto con las matrices de peso  $\mathbf{U}$ ,  $\mathbf{V}$  y  $\mathbf{W}$ . La función de activación es la *tangente hiperbólica*.

# Redes Recurrentes: Backpropagation through time (BPTT)

- Calcular el gradiente a través de una red neuronal recurrente es sencillo.
- No se necesitan algoritmos especializados.
- Los gradientes obtenidos por retropropagación pueden usarse con cualquier técnica basada en gradientes de uso general para entrenar una RNN.
- Los nodos de nuestro gráfico computacional incluyen los parámetros  $\mathbf{U}$ ,  $\mathbf{V}$ ,  $\mathbf{W}$ ,  $\mathbf{b}$  y  $\mathbf{c}$ , así como la secuencia de nodos indexados por  $t$  para  $\mathbf{x}^{(t)}$ ,  $\mathbf{h}^{(t)}$ ,  $\mathbf{o}^{(t)}$  y  $\mathbf{L}^{(t)}$ . Para cada nodo  $\mathbf{N}$ , necesitamos calcular el gradiente  $\nabla_{\mathbf{N}} \mathbf{L}$  recursivamente, con base en el gradiente calculado en los nodos que lo siguen en el gráfico.

# Redes Recurrentes: BPTT

- Comenzamos la recursión con los nodos que preceden inmediatamente a la pérdida final

$$\frac{\partial L}{\partial L^{(t)}} = 1$$

- En esta derivación, asumimos que las salidas  $\mathbf{o}^{(t)}$  se usan como argumento de la función *softmax* para obtener el vector  $\hat{\mathbf{y}}$  de probabilidades sobre la salida.
- También asumimos que la *pérdida* es la probabilidad logarítmica negativa del verdadero objetivo  $\mathbf{y}^{(t)}$  dada la entrada hasta el momento.

# Redes Recurrentes: BPTT

- El gradiente  $\nabla_{\mathbf{o}(t)} \mathbf{L}$  en las salidas en el paso de tiempo  $t$ , para todo  $i$ ,  $t$ , es el siguiente:

$$(\nabla_{\mathbf{o}(t)} L)_i = \frac{\partial L}{\partial o_i^{(t)}} = \frac{\partial L}{\partial L^{(t)}} \frac{\partial L^{(t)}}{\partial o_i^{(t)}} = \hat{y}_i^{(t)} - \mathbf{1}_{i,y^{(t)}}$$

# Redes Recurrentes: BPTT

- Continuando calculando el camino hacia atrás, comenzando desde el final de la secuencia.
- En el último paso de tiempo  $T$ ,  $\mathbf{h}^{(T)}$  solo tiene  $\mathbf{o}^{(T)}$  como descendiente, por lo que su gradiente es:

$$\nabla_{\mathbf{h}^{(T)}} L = \mathbf{V}^\top \nabla_{\mathbf{o}^{(T)}} L$$

# Redes Recurrentes: BPTT

- Podemos iterar hacia atrás en el tiempo para propagar hacia atrás los gradientes a través del tiempo, desde  $\mathbf{t}=\mathbf{T}-1$  hasta  $\mathbf{t}=\mathbf{1}$ , observando que  $\mathbf{h}^{(t)}$  (para  $\mathbf{t} < \mathbf{T}$ ) tiene como descendientes tanto  $\mathbf{o}^{(t)}$  como  $\mathbf{h}^{(t+1)}$ . Su gradiente viene dado por:

$$\begin{aligned}\nabla_{\mathbf{h}^{(t)}} L &= \left( \frac{\partial \mathbf{h}^{(t+1)}}{\partial \mathbf{h}^{(t)}} \right)^\top (\nabla_{\mathbf{h}^{(t+1)}} L) + \left( \frac{\partial \mathbf{o}^{(t)}}{\partial \mathbf{h}^{(t)}} \right)^\top (\nabla_{\mathbf{o}^{(t)}} L) \\ &= \mathbf{W}^\top (\nabla_{\mathbf{h}^{(t+1)}} L) \text{diag} \left( 1 - \left( \mathbf{h}^{(t+1)} \right)^2 \right) + \mathbf{V}^\top (\nabla_{\mathbf{o}^{(t)}} L)\end{aligned}$$

- donde la diagonal indica la matriz diagonal que contiene los elementos:  $1 - (h_i^{(t+1)})^2$ . Este es el [Jacobiano](#) de la tangente hiperbólica asociada con la unidad oculta  $\mathbf{i}$  en el tiempo  $\mathbf{t} + 1$ .

$$\nabla_{\mathbf{c}} L = \sum_t \left( \frac{\partial \mathbf{o}^{(t)}}{\partial \mathbf{c}} \right)^\top \nabla_{\mathbf{o}^{(t)}} L = \sum_t \nabla_{\mathbf{o}^{(t)}} L$$

$$\nabla_{\mathbf{b}} L = \sum_t \left( \frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{b}^{(t)}} \right)^\top \nabla_{\mathbf{h}^{(t)}} L = \sum_t \text{diag} \left( 1 - \left( \mathbf{h}^{(t)} \right)^2 \right) \nabla_{\mathbf{h}^{(t)}} L$$

$$\nabla_{\mathbf{V}} L = \sum_t \sum_i \left( \frac{\partial L}{\partial o_i^{(t)}} \right) \nabla_{\mathbf{V} o_i^{(t)}} = \sum_t (\nabla_{\mathbf{o}^{(t)}} L) \mathbf{h}^{(t)\top}$$

$$\begin{aligned} \nabla_{\mathbf{W}} L &= \sum_t \sum_i \left( \frac{\partial L}{\partial h_i^{(t)}} \right) \nabla_{\mathbf{W}^{(t)} h_i^{(t)}} \\ &= \sum_t \text{diag} \left( 1 - \left( \mathbf{h}^{(t)} \right)^2 \right) (\nabla_{\mathbf{h}^{(t)}} L) \mathbf{h}^{(t-1)\top} \end{aligned}$$

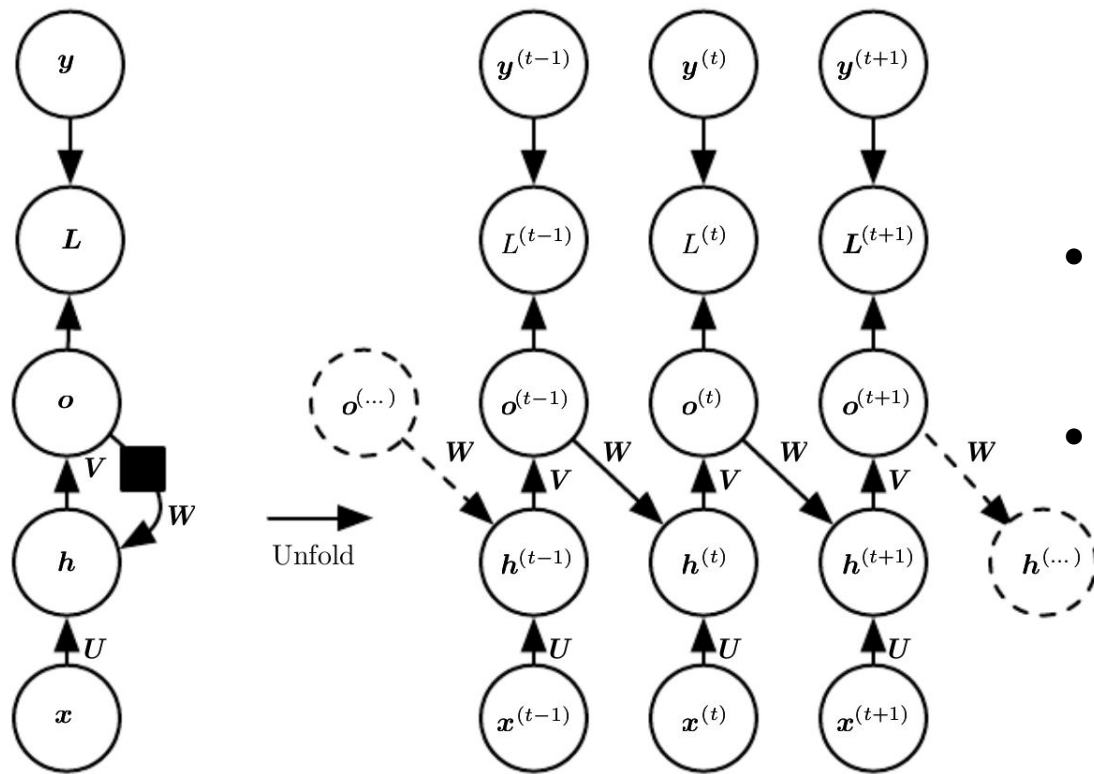
$$\begin{aligned} \nabla_{\mathbf{U}} L &= \sum_t \sum_i \left( \frac{\partial L}{\partial h_i^{(t)}} \right) \nabla_{\mathbf{U}^{(t)} h_i^{(t)}} \\ &= \sum_t \text{diag} \left( 1 - \left( \mathbf{h}^{(t)} \right)^2 \right) (\nabla_{\mathbf{h}^{(t)}} L) \mathbf{x}^{(t)\top} \end{aligned}$$

## RRN: BPTT

- Una vez que se obtienen los gradientes en los nodos internos del grafo, podemos obtener los gradientes en los nodos de los parámetros.
- No necesitamos calcular el gradiente con respecto a  $\mathbf{x}^{(t)}$  para el entrenamiento porque no tiene ningún parámetro como ancestros en el gráfico computacional que define la pérdida.

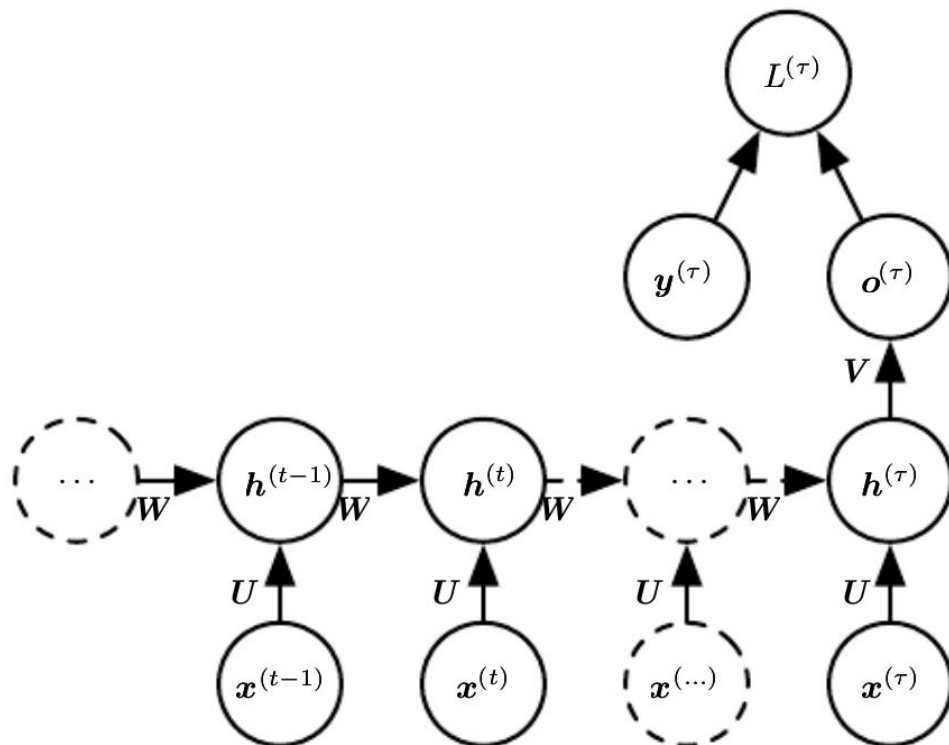


# RNN: Otras arquitecturas



- Redes recurrentes que producen una salida en cada paso de tiempo y tienen conexiones recurrentes sólo desde la salida en un paso de tiempo hasta las unidades ocultas en el siguiente paso de tiempo.
- A menos que  $o$  sea rico y de dimensiones muy altas, por lo general carecera de información importante del pasado.
- Esto hace que el RNN en esta figura sea menos poderosa, pero puede ser más fácil de entrenar porque cada paso de tiempo se puede entrenar de forma aislada de los demás, lo que permite una mayor paralelización durante el entrenamiento.

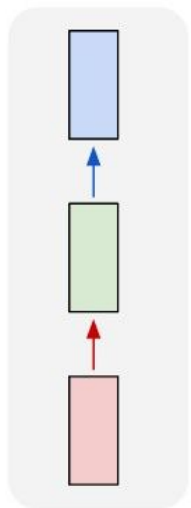
# RNN: Otras arquitecturas



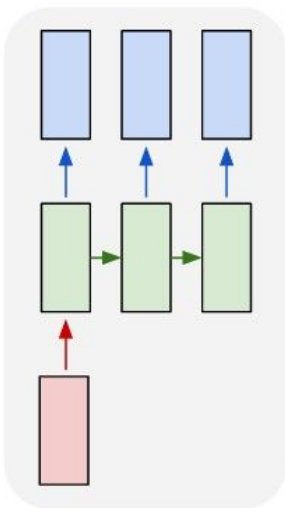
- Redes recurrentes con conexiones recurrentes entre unidades ocultas, que leen una secuencia completa y luego producen una única salida.
- Esta red se puede usar para resumir una secuencia y producir una representación de tamaño fijo que se usa como entrada para un procesamiento posterior.
- Puede haber un *target* ( $y$ ) justo al final (como se muestra aquí) o el gradiente en la salida  $o^{(t)}$  se puede obtener con el BPTT desde otros módulos posteriores.

# RNN: Otras arquitecturas

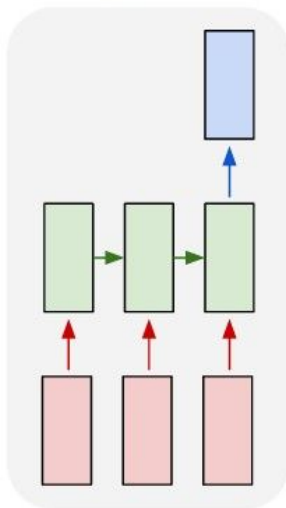
one to one



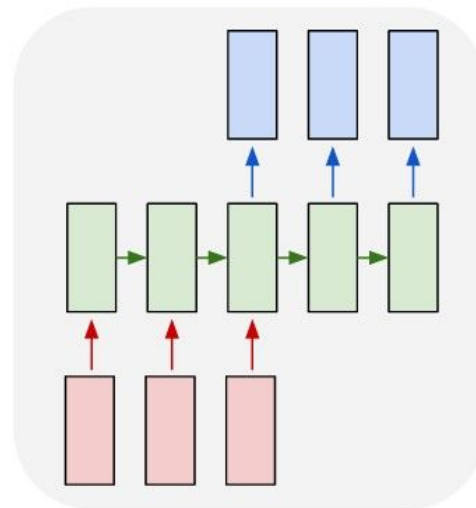
one to many



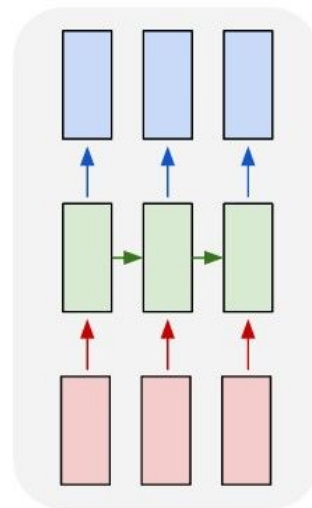
many to one



many to many



many to many

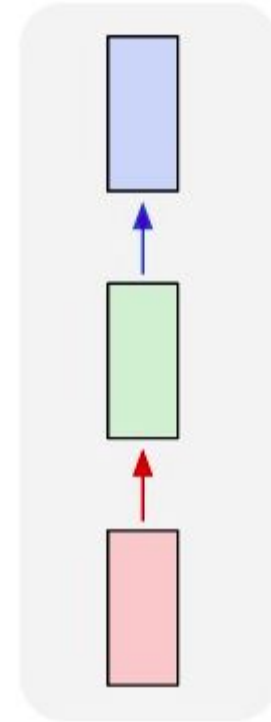


Cada rectángulo es un vector y las flechas representan funciones (por ejemplo, multiplicación de matrices). Los vectores de entrada están en rojo, los vectores de salida están en azul y los vectores verdes mantienen el estado de RNN

# RNN: Otras arquitecturas

- RNN con entrada de tamaño fijo hasta salida de tamaño fijo.
- Se utilizan, por ejemplo para: clasificación de imágenes.

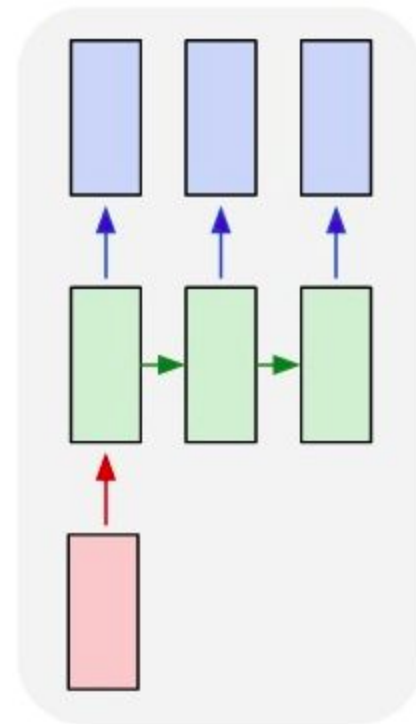
one to one



# RNN: Otras arquitecturas

- RNN con salida de secuencias a partir de una entrada
- Se puede emplear para: *image captioning*, toma una imagen y genera una oración de palabras para esa imagen.
- *Music generation*

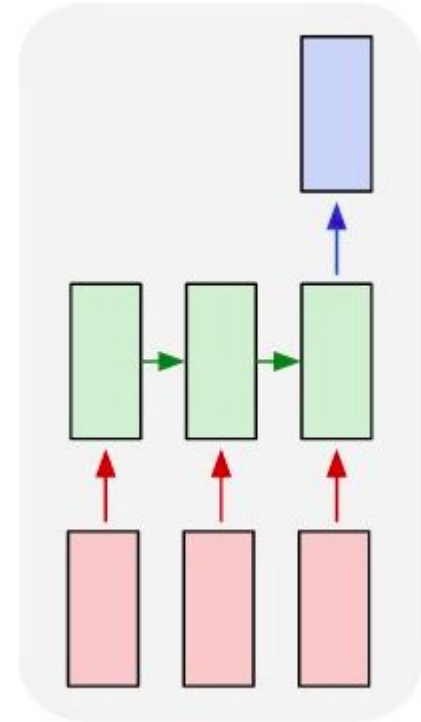
one to many



# RNN: Otras arquitecturas

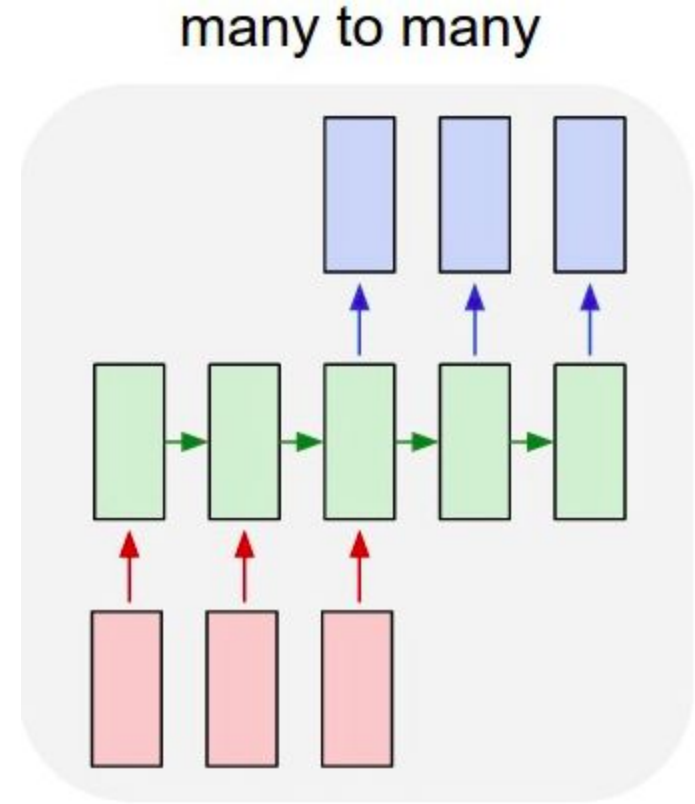
- RNN con varias entradas y solo una salida
- Se puede usar para el análisis de sentimientos, donde una oración determinada se clasifica como expresión de sentimiento positivo o negativo.

many to one



# RNN: Otras arquitecturas

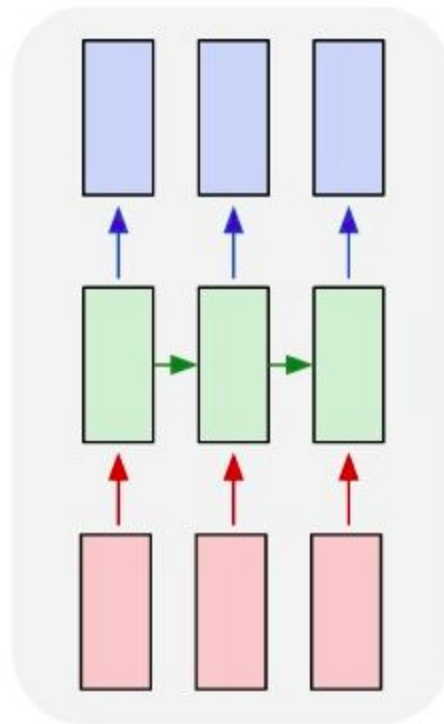
- RNN con múltiples entradas y salidas
- Se han empleado para traducción automática: un RNN lee una oración en inglés y luego genera una oración en francés.



# RNN: Otras arquitecturas

- RNN con entradas y salidas de secuencias sincronizadas
- Se pueden usar para clasificación de vídeo: donde deseamos etiquetar cada cuadro del video).
- También se utiliza para el reconocimiento de entidades (NER)
- Tenga en cuenta que en todos los casos no hay restricciones pre-especificadas en las secuencias de longitudes porque la transformación recurrente (verde) es fija y se puede aplicar tantas veces como queramos.

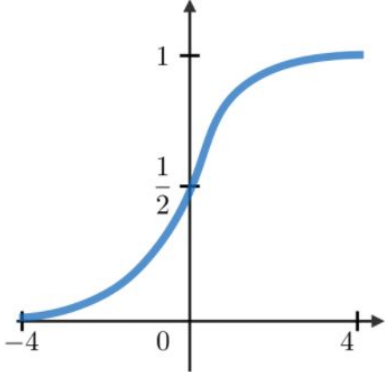
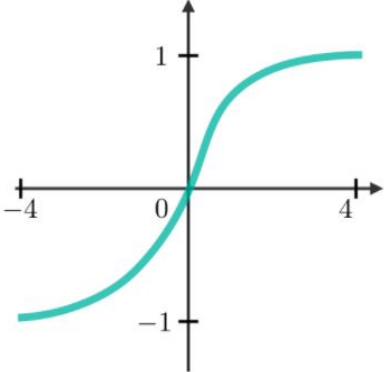
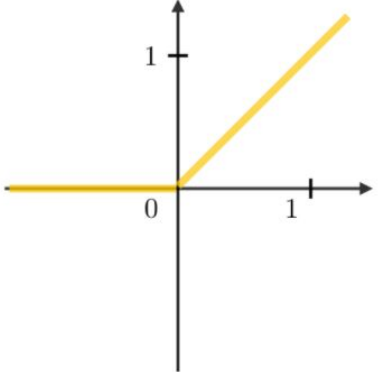
many to many





# Funciones de activación

Las funciones de activación más comunes utilizadas en los módulos RNN se describen a continuación:

Sigmoid	Tanh	RELU
$g(z) = \frac{1}{1 + e^{-z}}$	$g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	$g(z) = \max(0, z)$
		

# Ventajas y desventajas de una típica RNN

## VENTAJAS

- Posibilidad de procesar entradas de cualquier longitud
- El tamaño del modelo no aumenta con el tamaño de la entrada
- El cálculo tiene en cuenta la información histórica
- Los pesos se comparten a lo largo del tiempo

## DESVENTAJAS

- El cálculo es lento
- Dificultad para acceder a información de hace mucho tiempo
- No se puede considerar ninguna entrada futura para el estado actual