# Deep Learning

## Sequence Modeling: Recurrent and Recursive Nets

Resumen del capítulo 10
*Deep Learning,* Ian Goodfellow
www.deeplearningbook.org

Angel Navia Vázquez
angel.navia@uc3m.es

# Introducción

- Redes Neuronales Recurrentes (RNNs): familia de NNs para procesar datos secuenciales, de longitud variable (*análogo a redes convolucionales (CNNs) para datos en rejilla*)
- Se basan en la compartición de parámetros en el modelo, para favorecer la generalización
- Se modela la estructura interna de los datos, independientemente de su lugar de aparición (ej: procesado lenguaje natural)
- Primera aproximación: TDNN (Time Delay NN), una red convolucional 1-D, pero no es profunda, sólo toma información de los vecinos inmediatos

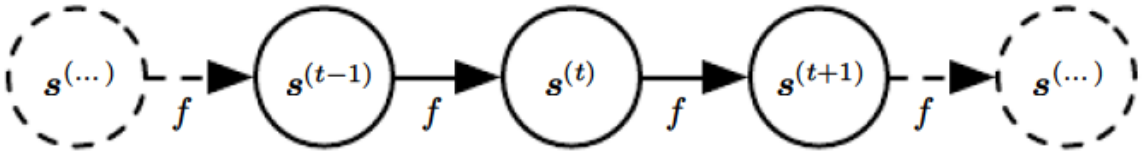
# RNNs

- La salida depende de salidas pasadas (ej. Filtros FIR vs. IIR)

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

- $s^{(t)}$ -> estado del modelo
- $\theta$ -> parámetros del modelo
- La recurrencia produce compartición de parámetros a través de un grafo muy profundo (grafos con ciclos)
- Este modelo se puede desdoblar, por aplicación recursiva:

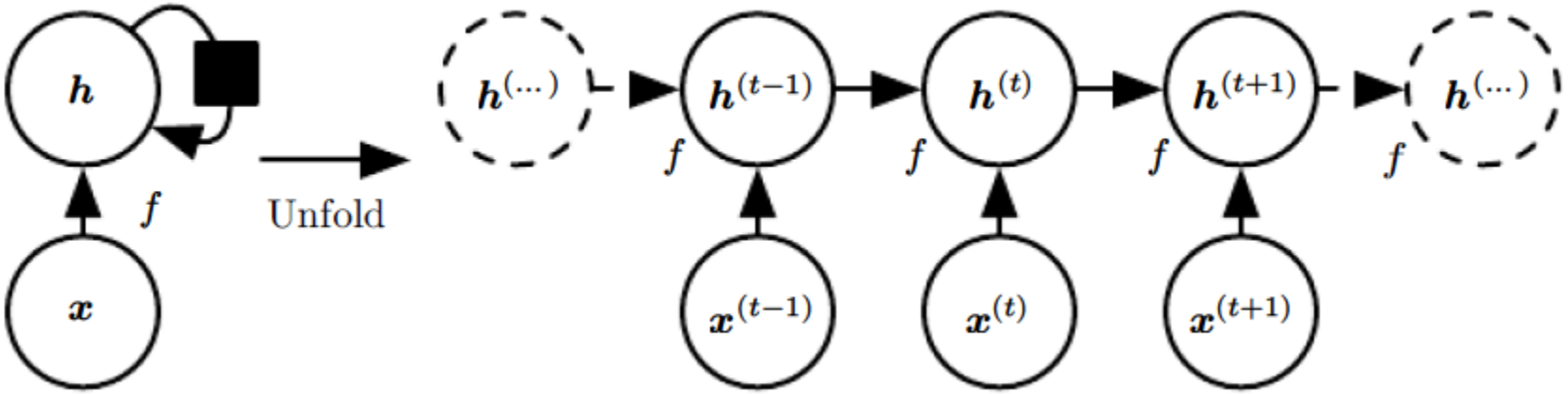
$$\begin{aligned}\boldsymbol{s}^{(3)} &= f(\boldsymbol{s}^{(2)}; \boldsymbol{\theta}) \\ &= f(f(\boldsymbol{s}^{(1)}; \boldsymbol{\theta}); \boldsymbol{\theta})\end{aligned}$$




- Desdoblamiento (“Unfolding”): concepto importante, permitirá definir diferentes algoritmos de aprendizaje

# Desdoblamiento de RNN sin salidas

- Ejemplo:




- Estado guardado en neuronas ocultas (hidden): **s** -> **h**
- En este caso, la entrada también influye:

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

- **h** almacena una "historia con pérdidas" de entradas pasadas, con énfasis en las relaciones más importantes para la tarea (ej. modelado lenguaje)

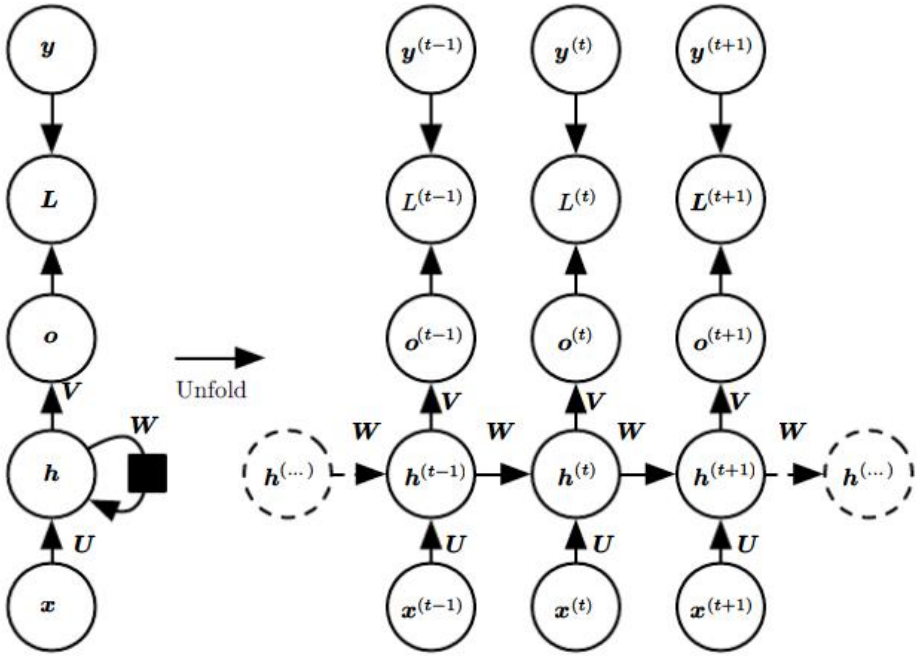
# Grafo directo vs. desdoblado

$$\begin{aligned}\boldsymbol{h}^{(t)} &= g^{(t)}(\boldsymbol{x}^{(t)}, \boldsymbol{x}^{(t-1)}, \boldsymbol{x}^{(t-2)}, \ldots, \boldsymbol{x}^{(2)}, \boldsymbol{x}^{(1)}) \\ &= f(\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)}; \boldsymbol{\theta}).\end{aligned}$$

- $h^{(t)}$ se puede ver como una función g() de todas las entradas pasadas
- El desdoblamiento permite factorizar g(.) como una aplicación repetida de f(.)
- Ventajas:
  - El modelo compacto siempre tiene el mismo tamaño de entrada, independientemente de la longitud de las señales
  - Se usa una misma función de transición f(), (compartición de parámetros -> mejor generalización)

# Tipos de RNNs (I)

- Recurrencia en neuronas ocultas, una salida en cada t (mapping input-output):

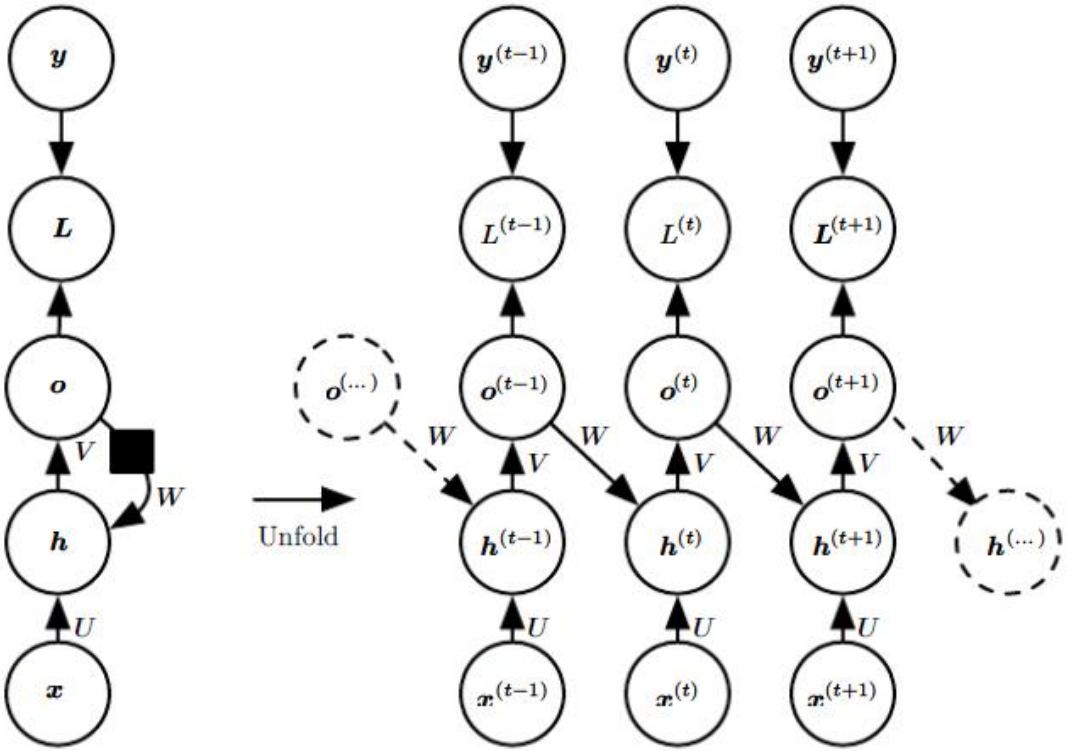



$$
\begin{aligned}
\boldsymbol{a}^{(t)} &= \boldsymbol{b} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)}, \\
\boldsymbol{h}^{(t)} &= \tanh(\boldsymbol{a}^{(t)}), \\
\boldsymbol{o}^{(t)} &= \boldsymbol{c} + \boldsymbol{V}\boldsymbol{h}^{(t)}, \\
\hat{\boldsymbol{y}}^{(t)}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

- Equivale a una máquina de Turing (si I/O es binario)
- Cálculo de gradientes es costoso (no paralelizable)
- BPTT (Back Propagation Through Time, ojo con ataduras)

# Tipos de RNNs (II)

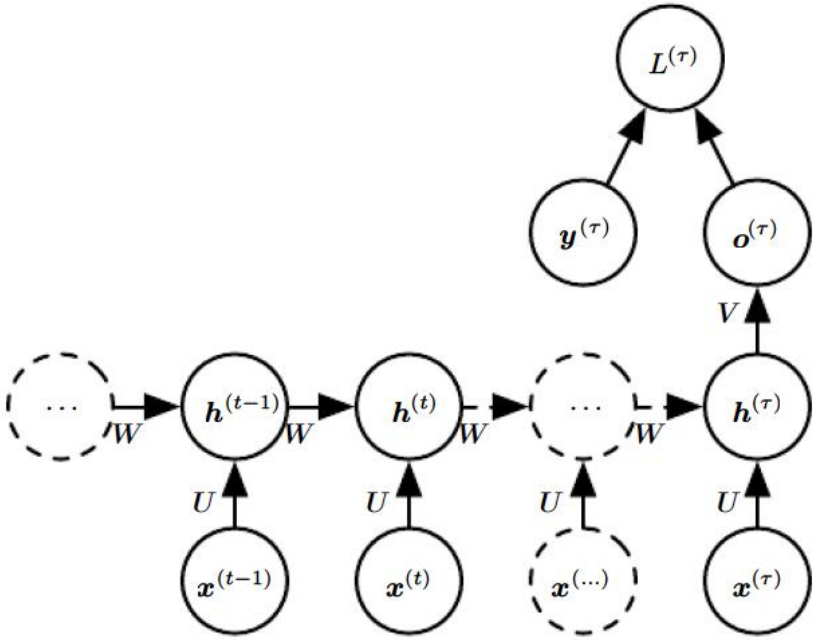
- Con recurrencia de salidas a ocultas , una salida en cada instante:




- Menos potente (dim o < dim h, menos inf. del pasado)
- Más fácil de entrenar (incluso paralelizable)

# Tipos de RNNs (III)

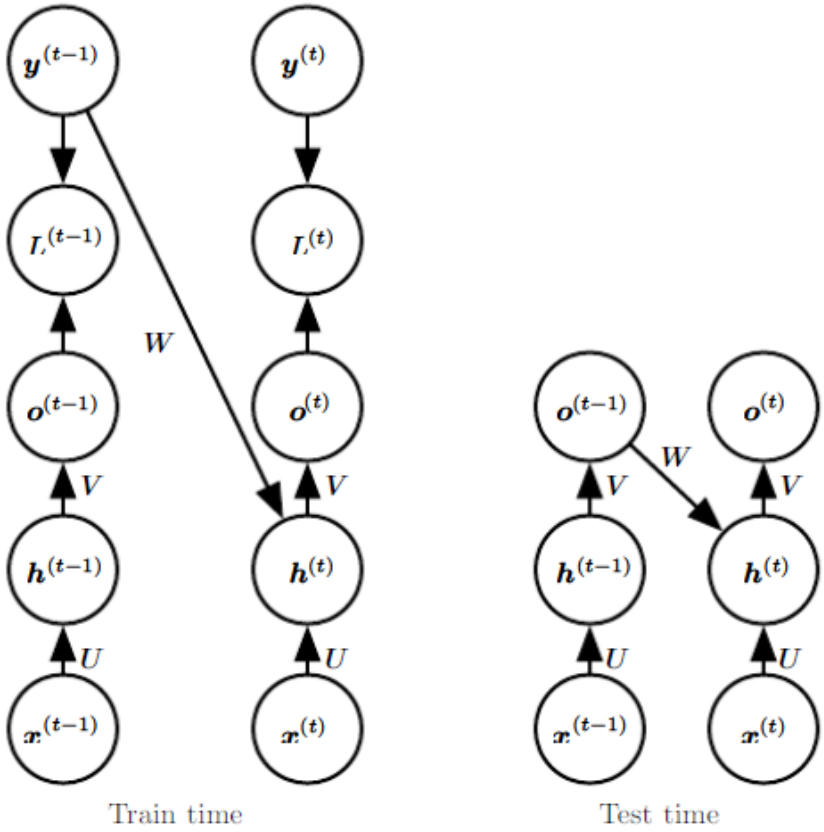
- Con recurrencia entre ocultas, una salida al final:




- Resume una secuencia en un valor final
- También requiere propagación recursiva de los gradientes, no paralelizable

# Teacher Forzing (TF)

- Durante entrenamiento se alimenta con las salidas correctas.
- Durante operación, la salida correcta se desconoce, se usa la estimación
- Es poco probable que se capture la dinámica completa de los datos (falla en bucle abierto)




- Opción 1: combinar varios pasos BPTT con TF
- Opción 2: usar gradualmente más datos generados y menos forzados

# RNNs como modelos gráficos dirigidos

- Con salida como DP, L = cross-entropía. Ej: MSE es la cross-entropía para unidades Gauss.
- Con objetivo log-verosimilitud, la RNN estima la distribución condicional:

$$\log p(\boldsymbol{y}^{(t)} \mid \boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(t)}) \qquad \log p(\boldsymbol{y}^{(t)} \mid \boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(t)}, \boldsymbol{y}^{(1)}, \ldots, \boldsymbol{y}^{(t-1)})$$

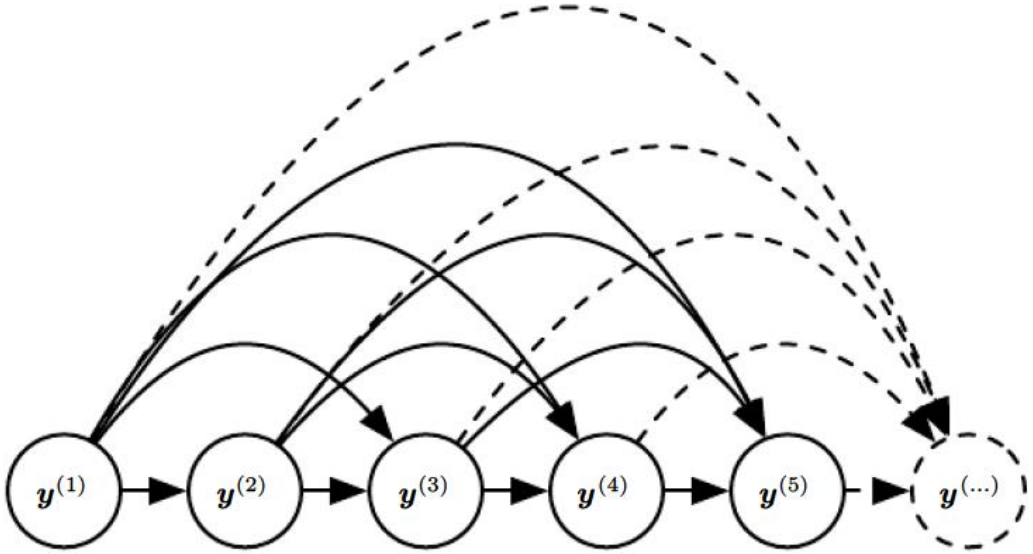
- Ejemplo: modelado de secuencia

$$P(\mathbb{Y}) = P(\mathbf{y}^{(1)}, \ldots, \mathbf{y}^{(\tau)}) = \prod_{t=1}^{\tau} P(\mathbf{y}^{(t)} \mid \mathbf{y}^{(t-1)}, \mathbf{y}^{(t-2)}, \ldots, \mathbf{y}^{(1)})$$

$$L = \sum_t L^{(t)} \qquad L^{(t)} = -\log P(\mathrm{y}^{(t)} = y^{(t)} \mid y^{(t-1)}, y^{(t-2)}, \ldots, y^{(1)})$$

# Modelo gráfico

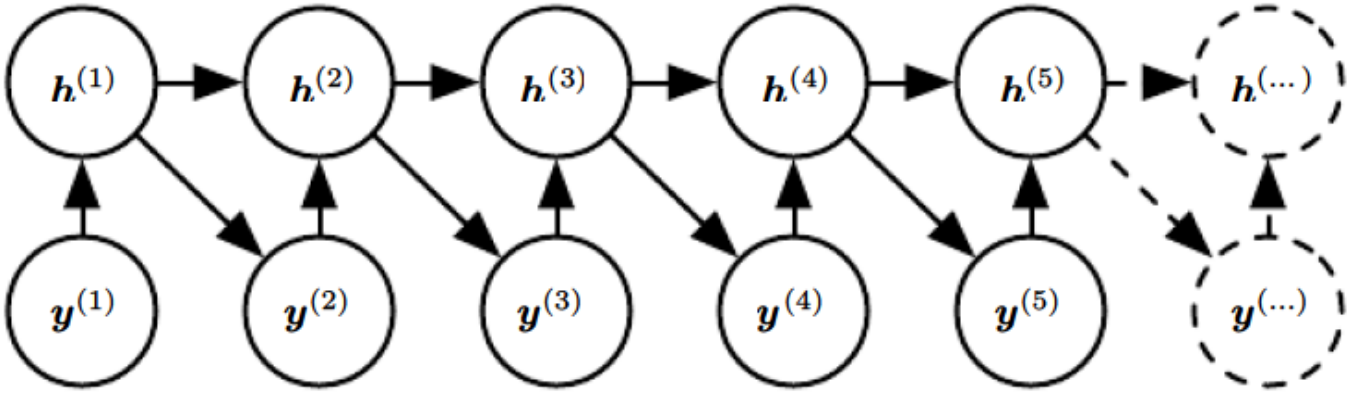
- Cada salida pasada influencia la salida actual




- La parametrización es ineficiente
- El modelo no es homogéneo (número variable de entradas)
- Equivale a marginalizar los estados ocultos

# RNN equivalente

- Usando un estado oculto (determinista):




- Parametrización más eficiente
- Modelo homogéneo (misma estructura)
- Se comparten pesos (asumiendo estacionariedad)
- La complejidad del modelo (h) se puede ajustar, pero no depende de la longitud de la entrada

# Modo generativo

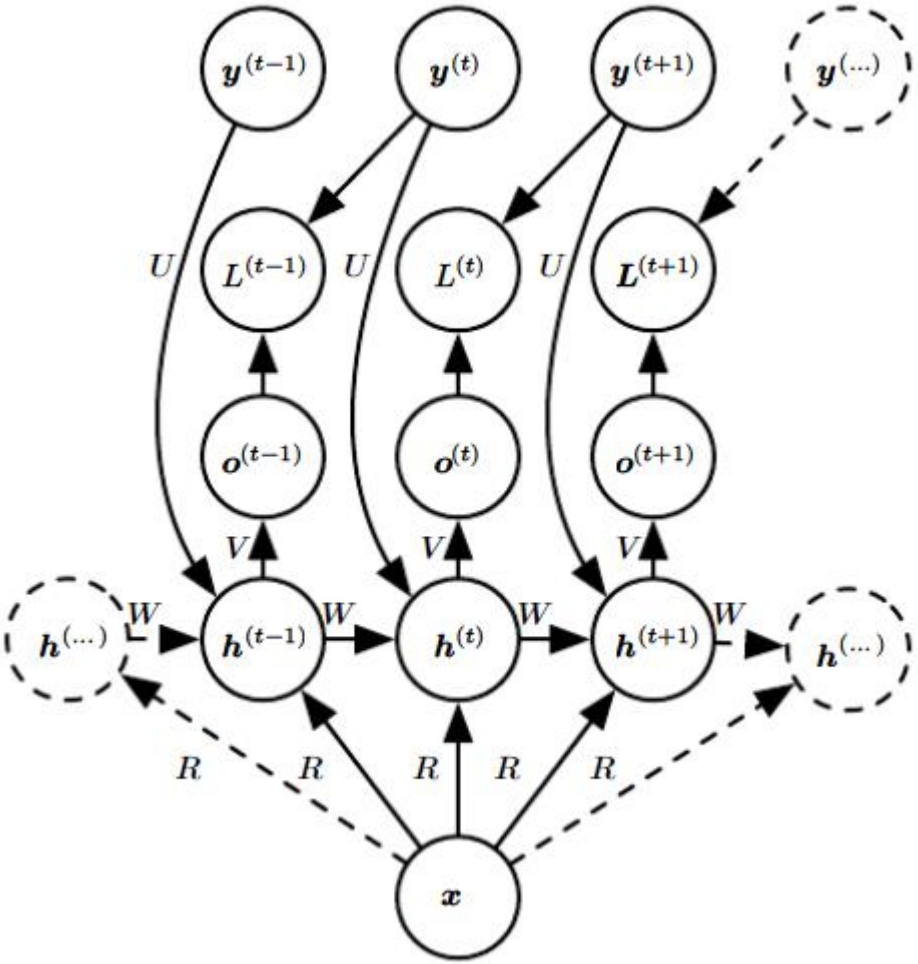
1. Muestrear de la distribución condicional en cada instante para obtener la salida
2. Determinar cuando parar de generar:
   - Si modelamos símbolos de un vocabulario, se puede usar un símbolo <end>
   - Introducir una salida Bernoulli adicional, que decide si parar o seguir (válido para valores no discretos)
   - Añadir una salida adicional que predice directamente la longitud de la secuencia

# Modelado con entrada única

- Se modela la distribución conjunta de las y, condicionada a la x.

- Ejemplo de uso: una

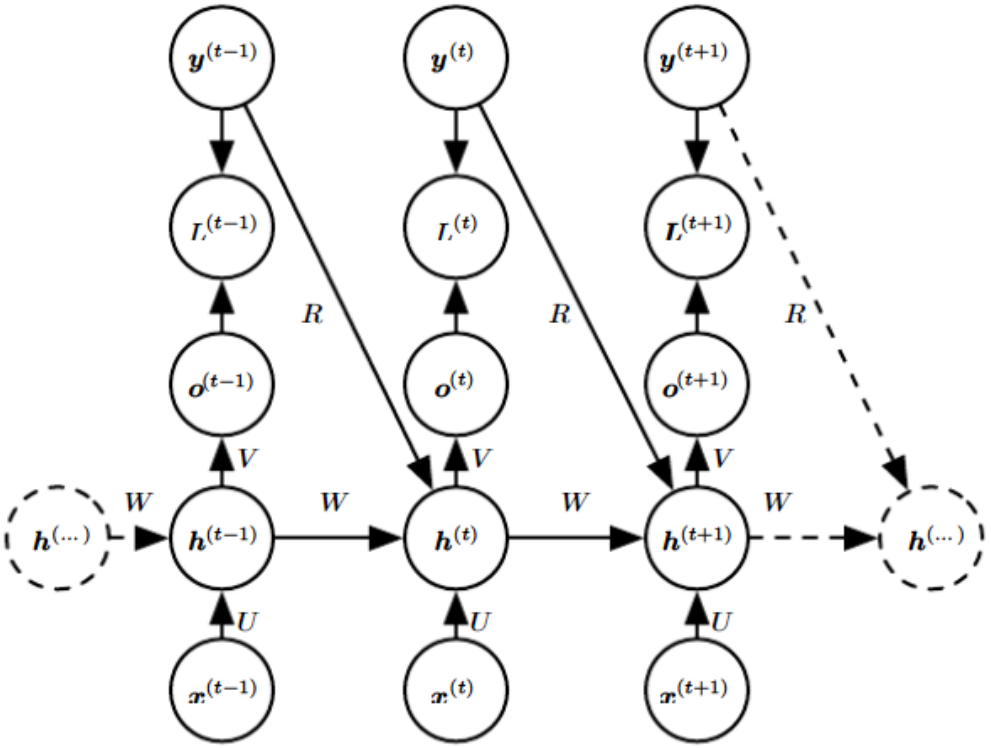
Imagen como entrada, la salida es una des-cripción textual de la misma.



# Modelado con entradas diferentes

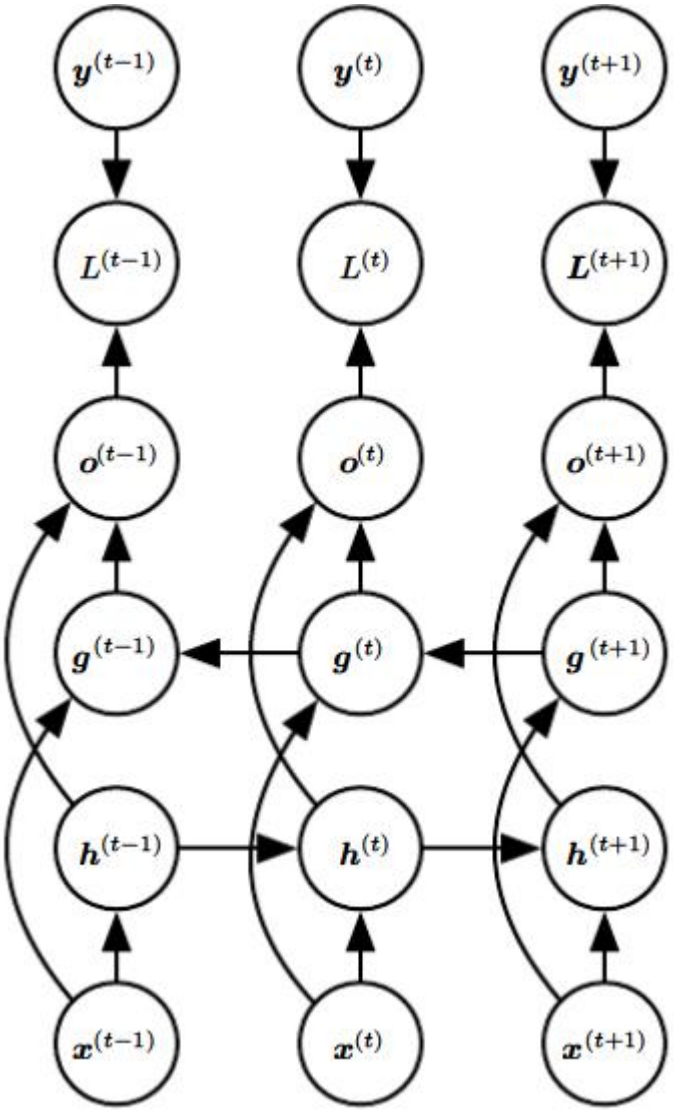
- Permite modelar

cualquier secuencia **y** dada cualquier se-cuencia **x** (ambas de la misma longitud)



# RNNs bidireccionales

- Hasta ahora, modelos causales
- Puede interesar usar pasado y futuro (rec. Voz (offline), análisis lingüístico, proc. imagen, etc.)
- Propagación progresiva (**h**) y regresiva (**g**)
- Extensible a bidimensional $h_{ij}$, $g_{ij}$



# Encoder-decoder (secuencias)

- RNN para mapear una secuencia de entrada a otra de salida (rec. voz, traducción, respuesta a preguntas)
- C = contexto, suele ser

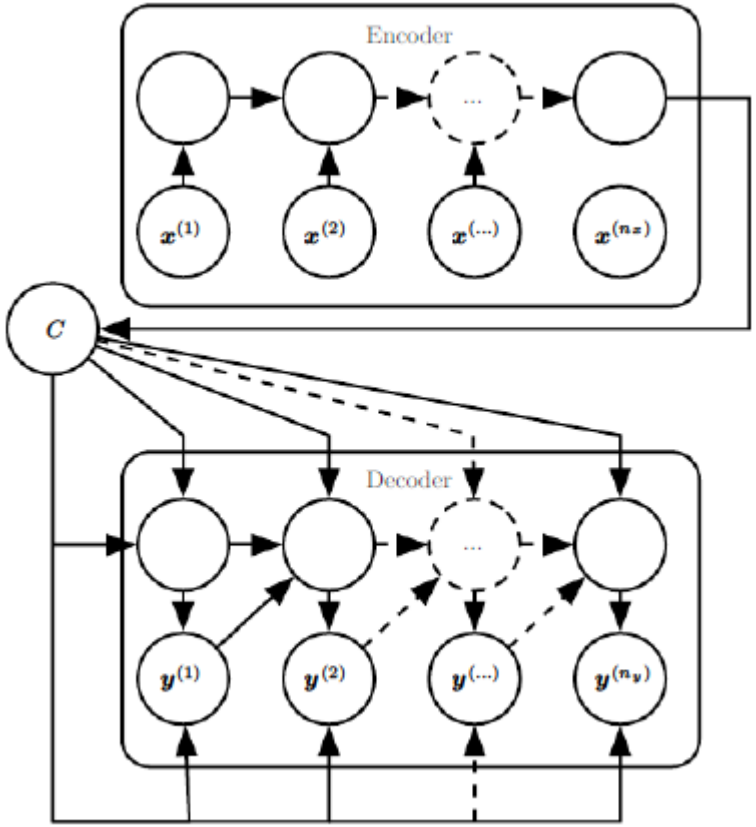
el último estado del encoder

- El encoder genera C a

partir de la entrada, y el

decoder genera la salida

a partir de C.



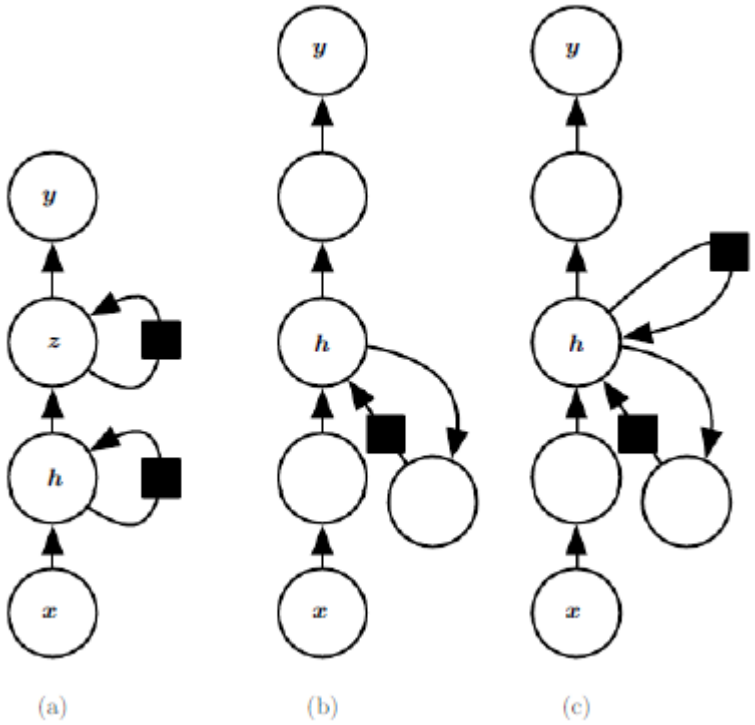
# RNNs profundas

- Se introduce profundidad en cada etapa (I->H, H-> O): hay ventajas en prestaciones:

a) el estado oculto se fragmenta en jerarquía

b) la recurrencia es un MLP

c) se acorta el camino mínimo de t a t+1 usando puenteos (favorece el entrenamiento)



# Red Neuronal Recursiva

- Su grafo se estructura como un árbol profundo, en vez de una cadena
- Se reduce el número de operaciones

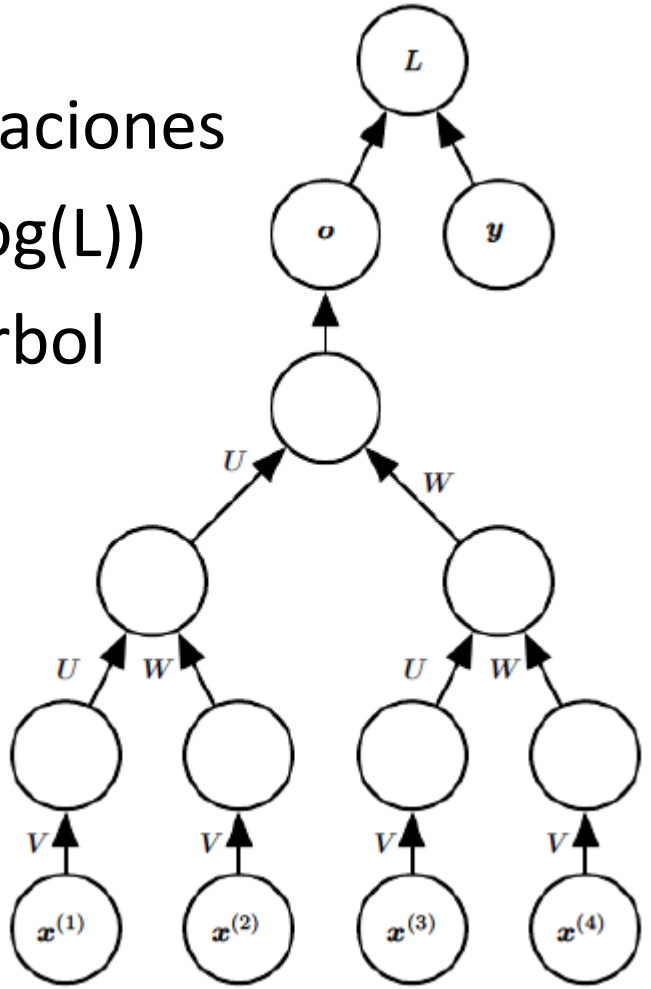
no lineales encadenadas (L -> log(L))

- La estructura puede ser un árbol

binario balanceado (genérico)

o el resultado de un parser

Lingüístico (específico)



# Dependencias a largo plazo (LT)

- Mismo problema que en redes profundas: los gradientes se desvanecen

$$\boldsymbol{h}^{(t)} = \left(\boldsymbol{W}^{t}\right)^{\top} \boldsymbol{h}^{(0)} \qquad \boldsymbol{h}^{(t)} = \boldsymbol{Q}^{\top} \boldsymbol{\Lambda}^{t} \boldsymbol{Q} \boldsymbol{h}^{(0)}$$

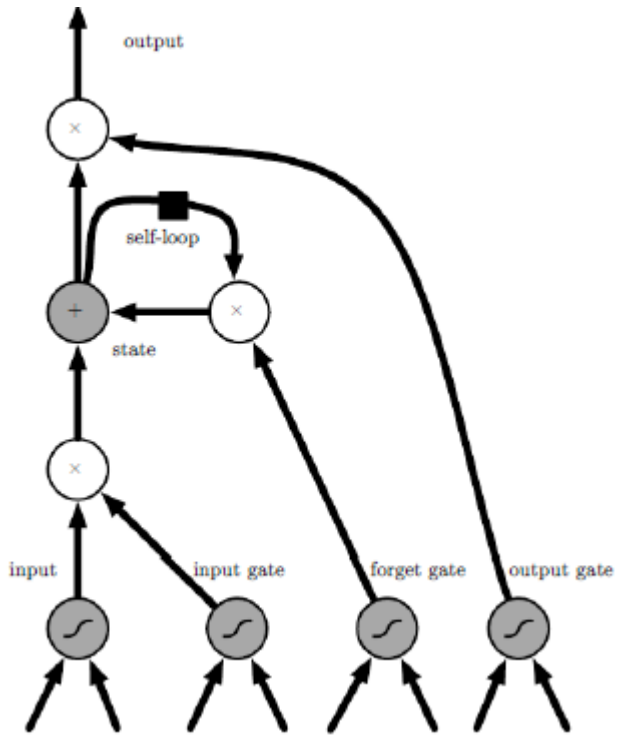
- En general: $\lambda^t$ los autovalores se desvanecen o crecen exponencialmente
- Para secuencias de longitud 20 o 30, la prob. de entrenar correctamente con SGD -> 0
- Quedarse en regiones del espacio donde los gradientes no se desvanecen no ayuda: para almacenar memorias de forma robusta hay que ir hacia zonas donde los gradientes precisamente se desvanecen
- Las dependencias a corto plazo (ST) dominan sobre las de largo plazo (LT), sus gradientes quedan enmascarados por los de ST

# Dependencias a largo plazo (II)

- Posibles soluciones:
  - Reservoir computing:
    - Echo-state networks (ESNs): fijar pesos recurrentes, sólo se entrenan los de salida (fácil). Similares a kernel que mapea secuencia a estado de duración finita.
    - (Liquid state networks (LSNs), análogas, con salidas binarias)
    - ¿cómo?: modelar como sistema dinámico en el límite de la estabilidad:
      - autovalores de J próximos a 1 ($g_{n+1} = J\, g_n$), caso lineal
      - Fijar radio espectral = 3, combinado con la saturación de las tanh, se estabiliza la salida en caso no lineal
    - Recientemente: entrenar capa recurrente con BPTT
  - Escala Temporal Múltiple: (modelo a diferentes escalas)
    - Skip connections: añadir conexiones directas desde variables en el pasado al presente (retardos mayores que 1)
    - Leaky Units: unidades con autoconexiones lineales y peso próximo a 1
    - Borrado de conexiones de corto alcance (por grupos de neuronas)
  - RNNs conmutadas: Long-Short Term Memory (LSTM). Generalizan los casos anteriores mediante el ajuste en cada instante de dichos pesos/caminos, incluso borrando estados una vez usados.

# Long Short-Term Memory (LSTM)

- Gran éxito en rec./gen. voz y escritura, traducción, descripción de imágenes o parsing.
- Crear caminos (loops) en que derivadas no desaparecen ni explotan, el gradiente se mantiene, todo controlado por el contexto
- La escala de tiempo es dinámica, controlada por otras unidades
- Las neuronas LSTM (recurrencia interna) sustituyen a las neuronas ocultas, con recurrencia externa adicional
- LSTM aprenden dependencias a largo plazo más fácil que las RNN clásicas
- Obtienen prestaciones de estado del arte en muchas tareas
- Otras variantes: GRUs, una única unidad controla el olvido y actualización




"neurona" LSTM

Gracias!!