Redes Neuronales Recurrentes en Pytorch

Diego Andrade Canosa

Roberto López Castro

Índice

- Introducción
- Conceptos básicos de RNNs
- RNNs en Pytorch

Introducción

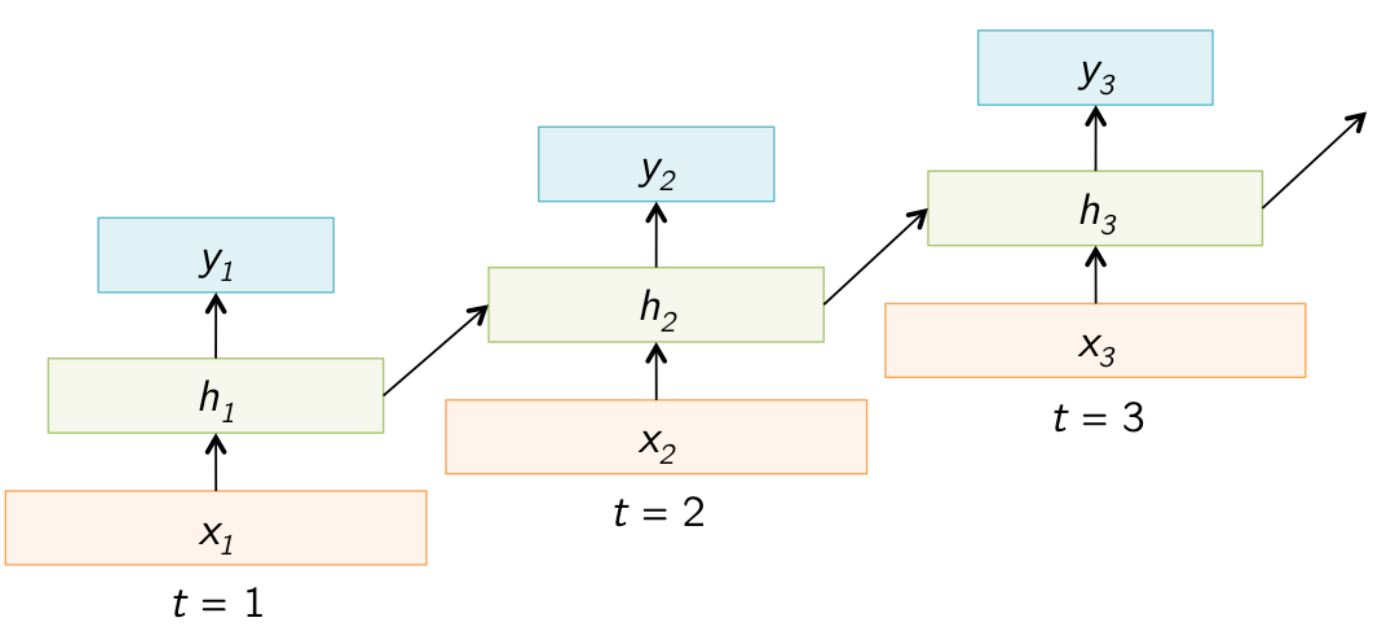
- Las Redes Neuronales Recurrentes (RNNs) son un tipo especial de RNs
  - Permiten procesar secuencias temporales de longitud variable
  - Proporcionan un mecanismo para almacenar y procesar información del contexto
  - La entrada se va suministrando secuencialmente
- Para ello, las RNNs crean ciclos entre los nodos de la red
  - Las entradas de un nodo reciben como entrada, también, la salida previa (como contexto)
  - Los valores intermedios (estado) almacenan información de las entradas previas

Introducción

- Aplicaciones
  - Predicción de redes temporales
  - Reconocimiento de voz
  - Traducción automática
  - Generación de texto

Introducción

Ilustración del concepto de RNN

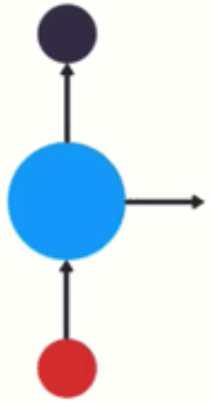


citic

Conceptos básicos de RNNs

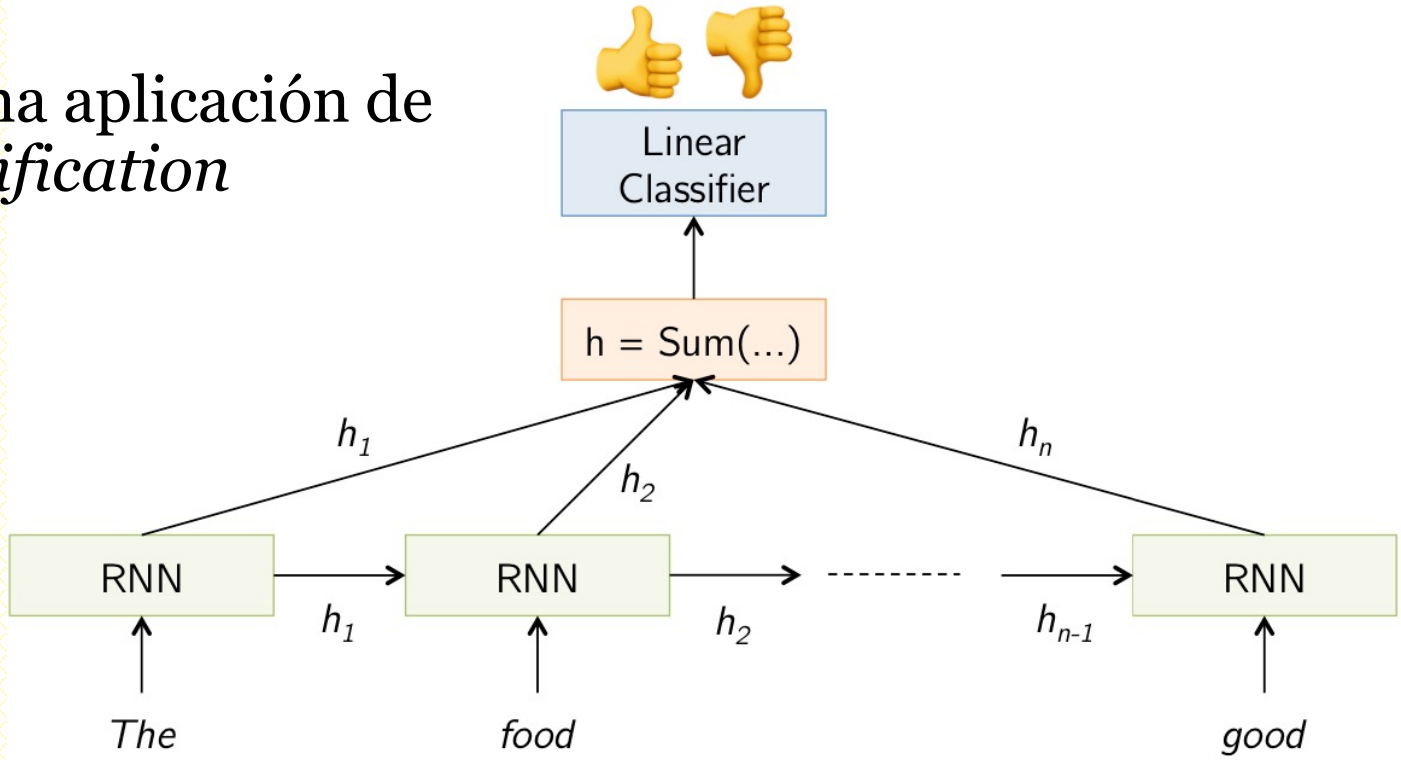
Conceptos básicos de RNNs

- Se comparten los pesos en el tiempo
- Se hacen copias de la misma *cell* en el tiempo con diferentes entradas como pasos de tiempo

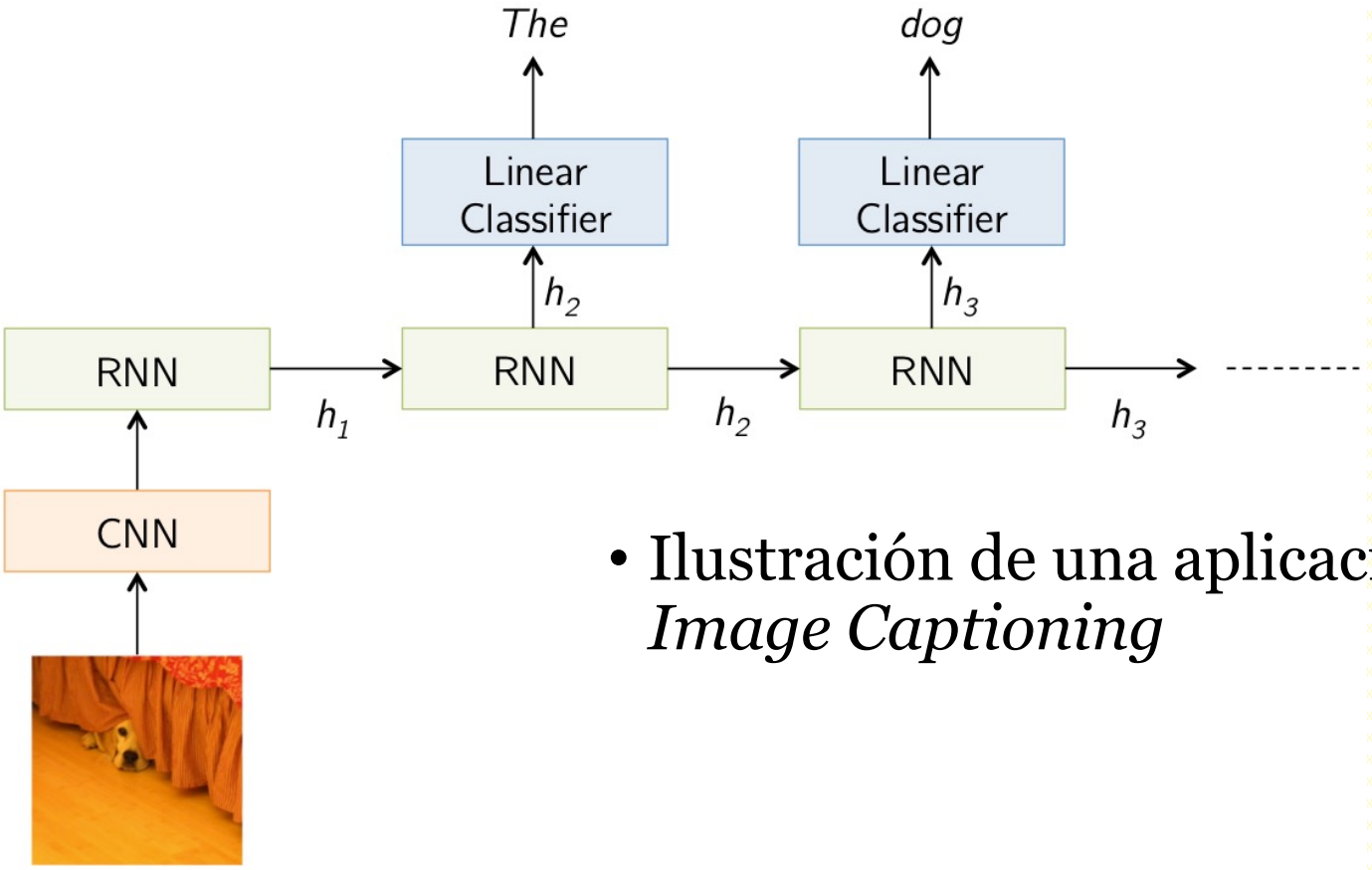


Conceptos básicos

- Ilustración de una aplicación de *Sentiment Classification*



Conceptos básicos

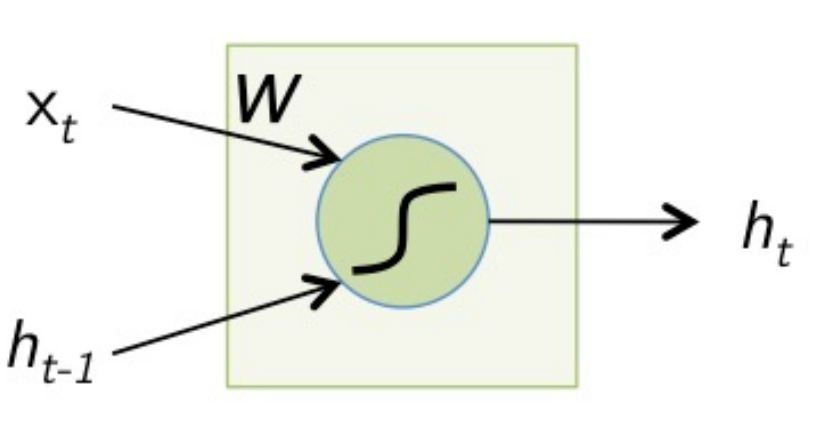


- Ilustración de una aplicación de *Image Captioning*

Conceptos básicos de RNNs: Vanilla

- La forma más básica de bloque constructor de una RNN es la *Vanilla Cell*

$$h_t = \tanh W \begin{bmatrix} x_t \\ h_{t-1} \end{bmatrix}$$



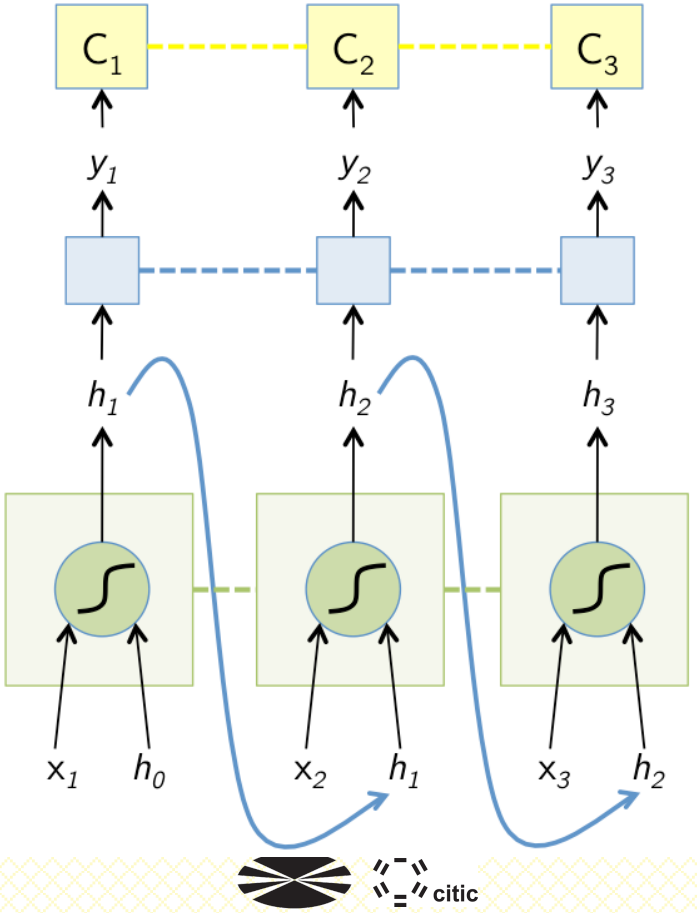
Conceptos básicos de RNNs: Vanilla *Cell*

$$h_t = tanhW[{}^{x_t}_{h_{t-1}}]$$

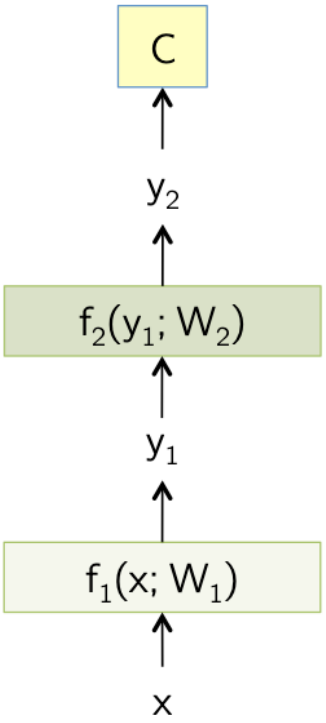
$$y_t = F(h_t)$$

$$C_t = Loss(y_t, GT_t)$$

------ Indica pesos compartidos

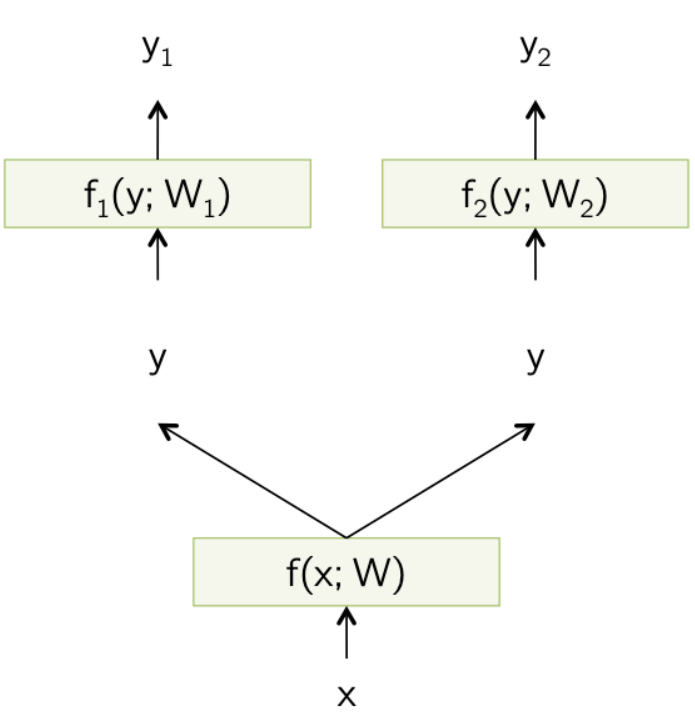


Conceptos básicos de RNNs: Vanilla



- Se pueden combinar múltiples capas

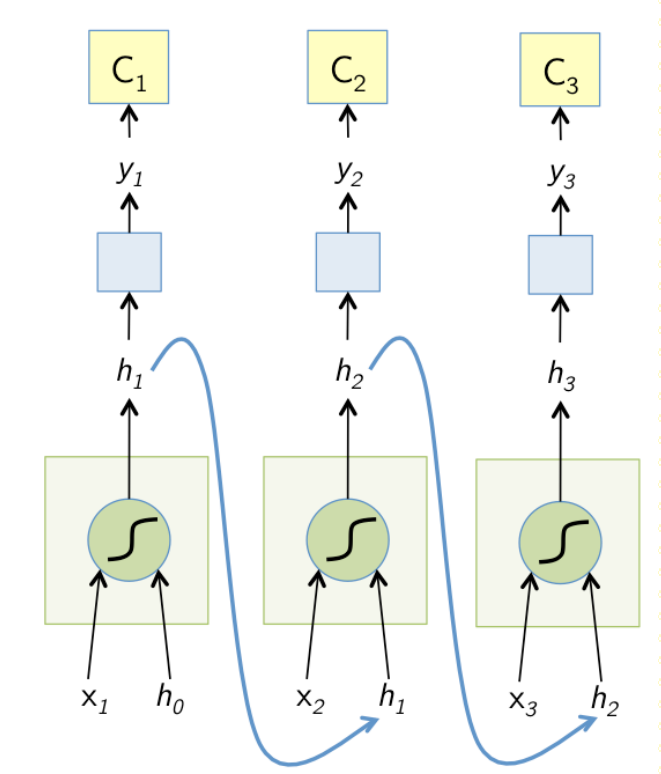
Conceptos básicos de RNNs: Vanilla



- Se pueden combinar múltiples capas
- También en forma de grafos

citic

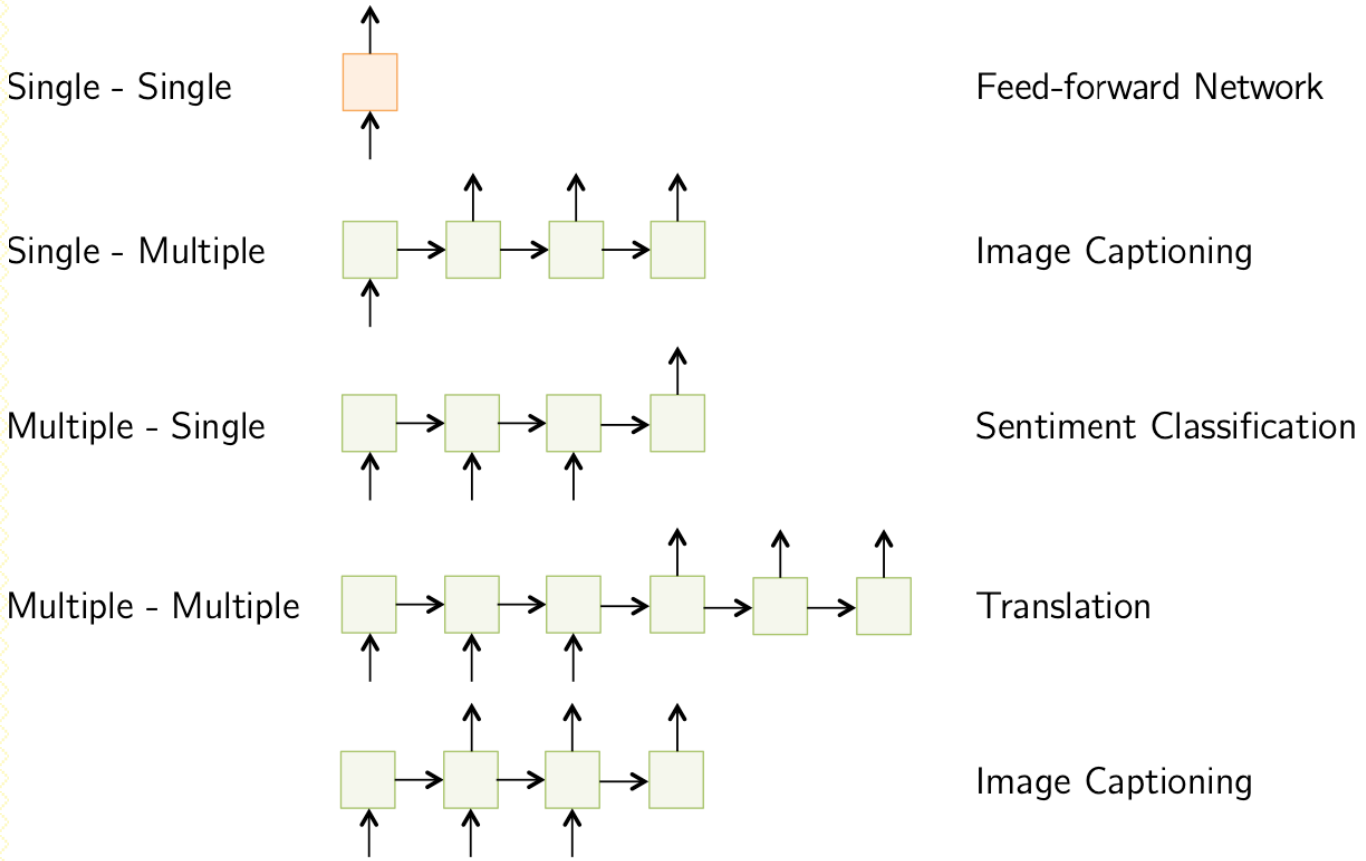
Conceptos básicos de RNNs: Vanilla Cell



- La versión *unfolded* es una red en la que las celdas están replicadas en el mismo paso de tiempo
- Toma toda la entrada al mismo tiempo

citic

Conceptos básicos: Escenarios de salida



citic

Conceptos básicos: Escenarios de Salida

- También se usa la notación:
  - One-to-one
  - One-to-many
  - Many-to-one
  - Many-to-many
  - Many-to-many

Problema del *vanishing gradient*

- La pasada *backward* en RNNs se realiza a través de distintos pasos de tiempo
  - Se calcula el gradiente en cada paso de tiempo
  - Si el efecto en la capa anterior es pequeño -> el gradiente calculado también será pequeño, y viceversa
    - Si encadenamos otro paso de tiempo, el gradiente calculado será todavía má pequeño
  - Produce un efecto por el cual los gradientes van disminuyendo en el tiempo (hacia atrás)
    - Es un problema para secuencias largas (compuestas de muchos pasos de tiempo)
      - No tiene memoria a largo plazo
    - A medida que vamos más hacia atrás los gradientes son más pequeños -> tienen menos influencia en la actualización de los pesos

Problema del *vanishing gradiente*

- LSTM y GRU plantean dos evoluciones distintas de las *Vanilla Cell* para implementar un mecanismo de memoria a largo plazo

Mi gato es … estaba enfermo

Mis gatos están … estaban enfermos

Para predecir estaba/estaban debe recordar el sujeto de la oración (Mi(s) gato(s))

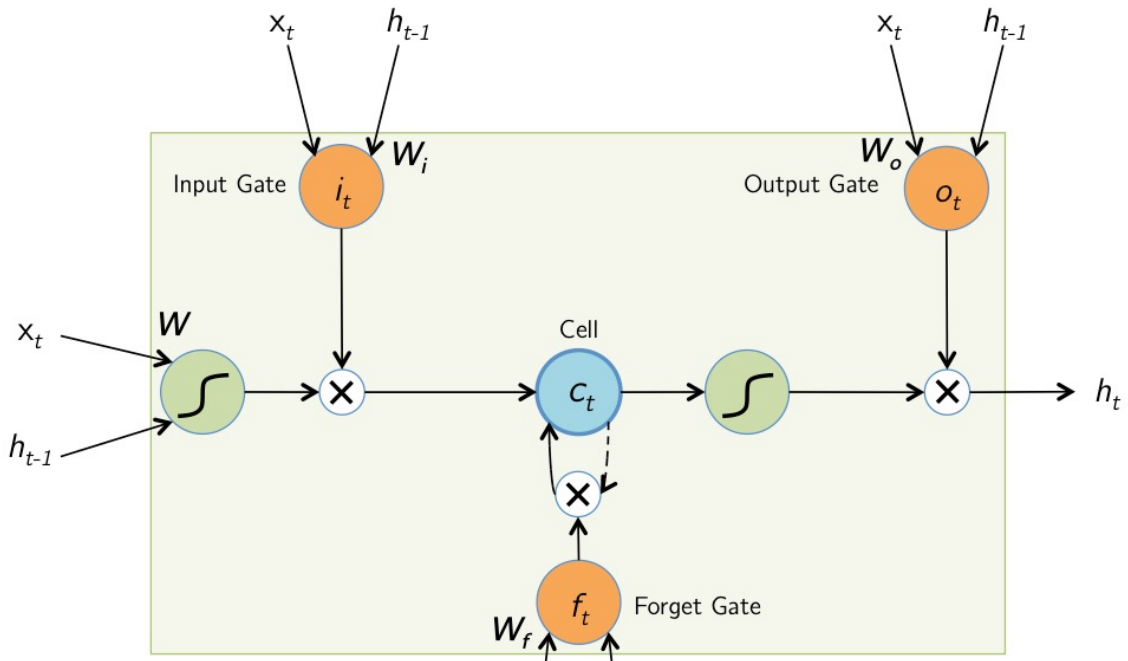
Conceptos básicos: LSTM

- Un problema común de arquitecturas de RNN como la *Vanilla* es que su rendimiento empeora cuando procesan secuencias largas
  - Son ineficaces reteniendo la memoria
  - Produce el problema de decaída de gradientes
- Para solucionarlo surgen alternativas como Long-Short Term Memory
  - Introduce el concepto de *Memory Cell*
    - Encapsula una arquitectura de 4 capas que interactúan de una manera determinada
  - Utiliza el concepto de *Constant Error Flow* para crear un *Constant Error Carousel* (CEC)
    - Asegura que no decae el gradiente al evitar que la información previa se propague por la red sin ser operados por los pesos que se encuentra, evitando la decaída de los gradientes

Conceptos básicos: LSTM

- El *Cell state* viaja por la cadena de unidades LSTM con una interacciones mínimas
  - Mejora la conservación de la memoria respecto a diseños previos
- Introduce el concepto de puerta (*gate*). Cada unidad LSTM tiene 3 tipos de puertas:
  - *Forget gate:* Decide qué información se deberían mantener o conservar en base a la entrada actual y el estado previo
  - *Input gate:* Actualiza el estado utilizando la nueva información en base a la entrada actual y el estado previo
  - *Output gate:* Decide cuál debería ser el siguiente estado de la celda en base a la entrada actual y el estado previo

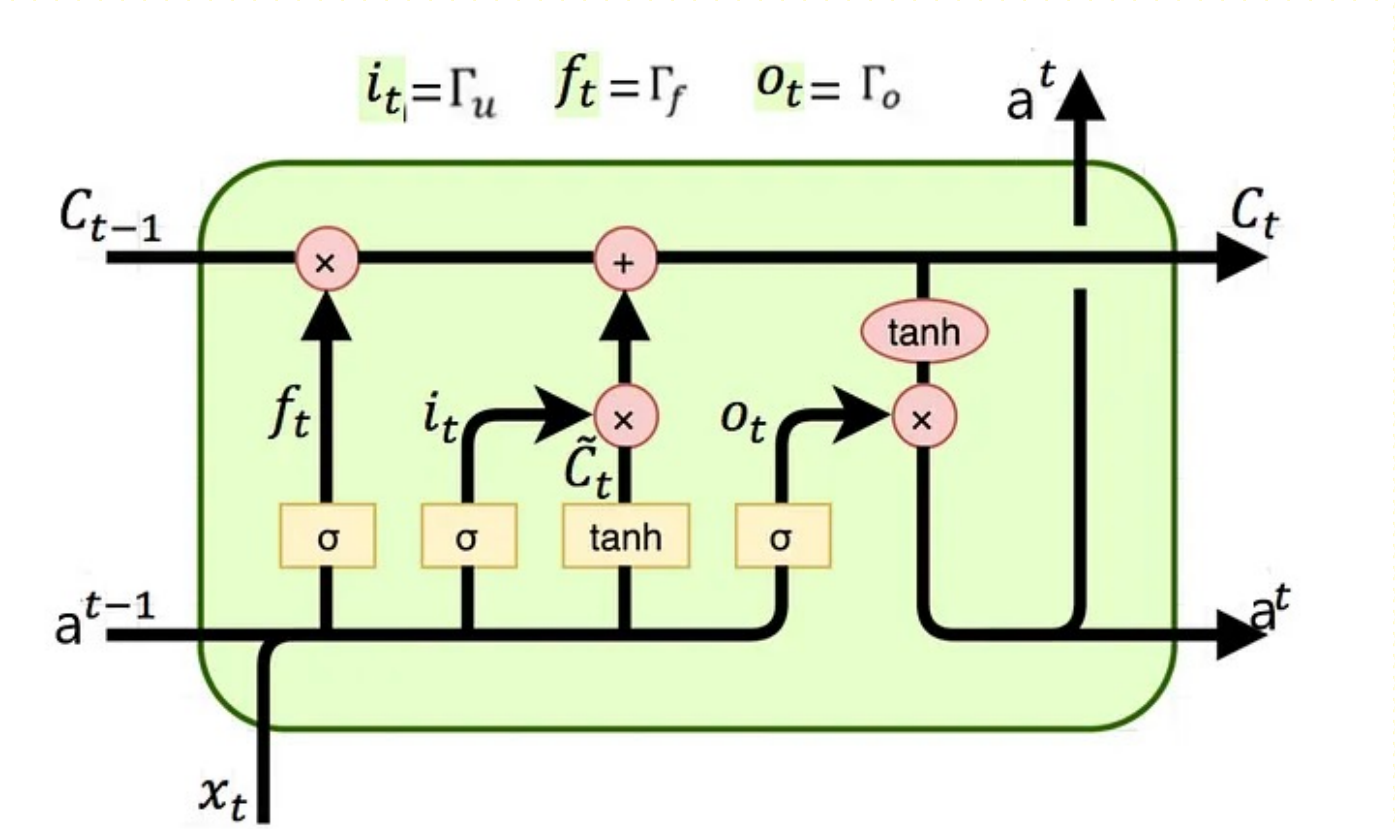
Conceptos básicos: LSTM



$$c_t = f_t \otimes c_{t-1} + i_t \otimes \tanh W \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix} \qquad f_t = \sigma\left(W_f \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix} + b_f \right)$$

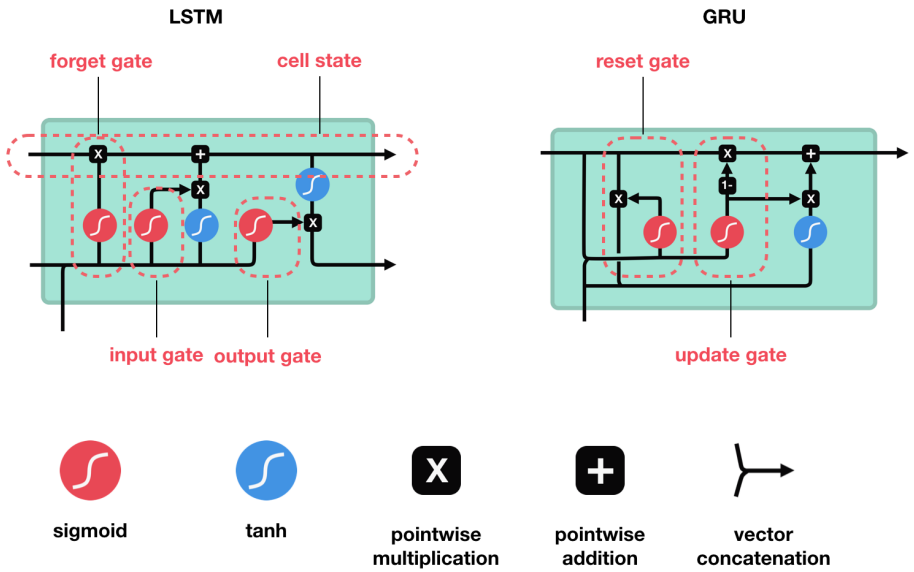
citic

Conceptos básicos: LSTM



Redes GRU

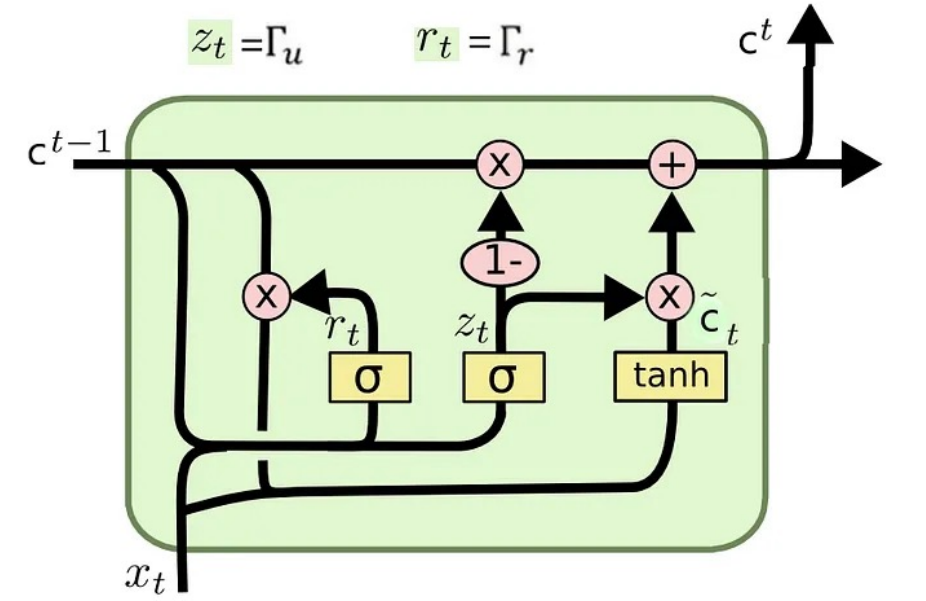
- GRU (*Gated Recurrent Unit: Rede*): Redes recurrentes introducidas en 2014 para solventar el problema de la desaparición del gradientes (*vanishing gradient*)
- Añaden 2 puertas al diseño básico de una RNN:
  - Update: Determina cuánta información previa se debe transmitir al siguiente paso de tiempo
  - Reset: Determina cuánta información no debería transmitirse al siguiente paso de tiempo
- Ambas puertas se codifican como vectores



citic

Redes GRU

- GRU (*Gated Recurrent Unit: Rede*): Redes recurrentes introducidas en 2014 para solventar el problema de la desaparición del gradientes (*vanishing gradient*)
- Añaden 2 puertas al diseño básico de una RNN:
  - Update: Determina cuánta información previa se debe transmitir al siguiente paso de tiempo
  - Reset: Determina cuánta información no debería transmitirse al siguiente paso de tiempo
- Ambas puertas se codifican como vectores



Redes GRU

- Update Gate: $z_t = \sigma(W_z \times x_t + U_z \times h_{t-1} + b_z)$
- Reset Gate: $r_t = \sigma(W_r \times x_t + U_r \times h_{t-1} + b_r$
- Candidate activation: $\grave{h_t} = \tanh(W \times x_t + U \times (r_t \circledast h_{t-1}) + b)$
- Salida final: $h_t = (1 - z_t) \circledast h_{t-1} + z_t \circledast \grave{h_t}$

σ es la función sigmoid

tanh es la tangente hiperbólica

$\circledast$ multiplicación elemento-a-element

x_t es la entrada en el paso de tiempo t

h_t es el estado oculto (en t)

W, *U y b* son pesos y bias

W son los pesos asociados con las entradas

U son los pesos asociados con los estados ocultos

RNNs en Pytorch

RNNs en Pytorch

- En el paquete torch.nn tenemos un conjunto completo de capas para RNNs
  - RNNs: Elman Muticapa
  - GRU: Gated Recurrent Unit
  - LSTM

RNNs en Pytorch

`nn.RNNBase`	
`nn.RNN`	Applies a multi-layer Elman RNN with $\tanh$ or ReLU non-linearity to an input sequence.
`nn.LSTM`	Applies a multi-layer long short-term memory (LSTM) RNN to an input sequence.
`nn.GRU`	Applies a multi-layer gated recurrent unit (GRU) RNN to an input sequence.
`nn.RNNCell`	An Elman RNN cell with tanh or ReLU non-linearity.
`nn.LSTMCell`	A long short-term memory (LSTM) cell.
`nn.GRUCell`	A gated recurrent unit (GRU) cell

Caso de estudio: LSTM para IMDB

- El dataset de IMDB (Internet Movie Database) es uno de los más populares para *Sentiment Analysis*
  - Compuesto de opiniones (50k) de los usuarios de imdb sobre películas
    - Cada opinión está etiquetada como positiva o negativa
  - Cada registro contiene el texto a clasificar y un entero indicando si es positivo o negativo
- *Sentiment analysis:* Aplicaciones que clasifican el sentimiento que denota un texto
  - En este caso positivo o negativo
    - Pero podría haber más categorías (ej. neutral)

Caso de estudio: LSTM para IMDB

- Las redes Long-Short Term Memory (LSTM) tienen en el *Sentiment Analysis* de texto una de sus aplicaciones más populares
- En Pytorch el dataset de imdb se puede cargar de forma sencilla a través de la librería torchtext
  - La carga requiere además un preprocesado de los datos
    - Tokenización del texto: Subdivisión del texto en tokens (palabras, subpalabras o incluso caracteres)
    - Creación de un *Vocab*: Encontrar cuántos tokens distintos aparecen en el dataset y asociarla a cada uno una representación numérica única
    - Convertir las entradas de texto en representaciones numéricas usando el *vocab* generado
    - Cada token del texto se convierte en un valor numérico

Caso de estudio: LSTM para IMDB

- La arquitectura de red podría estar compuesta de:
  - Una capa de *embedding*
  - Varias capas ocultas LSTM
  - Una capa Lineal final

Caso de estudio: LSTM para IMDB

- Una capa de Embedding
  - Procesa los datos de salida, transformándolos en un vector de valores
  - Las capas Embedding transforman un único valor de una variable discreta categórica a un vector de valores continuos
  - Se trata de una representación que se aprende durante el proceso de aprendizaje
  - En el caso de IMDB, representaremos cada texto de una review en un vector de *n* valores continuos, donde *n* es la longitud de la salida del Embedding aprendido.
  - La proximidad en la representación obtenida por dos textos (su vector embedded), debería representar (o capturar) la proximidad entre dichos textos

Caso de estudio: LSTM para IMDB

- Capas ocultas LSTM
  - Son varias capas encadenadas
    - La primera recibe como entrada la representación obtenida en la capa *Embedding*
    - Las capas ocultas se encadenan unas a otras modelando el problema a través del aprendizaje de sus parámetros
    - La última proporciona su salida como entrada a la página lineal

Caso de estudio: LSTM para IMDB

- Capa Lineal
  - Es una capa totalmente conectada que convierte la salida de la última capa oculta en uno o varios valores numéricos
  - En nuestro caso, ya que la red tiene que determinar si el *sentiment* del texto es positivo o negativo, basta con generar un único valor de salida

Caso de estudio: LSTM para IMDB

- El tamaño del *vocab* es 20000
- El *batch size* agrupará el conjunto de entrenamiento en lotes para ser procesados de forma eficiente
  - Debemos tener cuidado con no incrementarlo demasiado
    - Podría desbordar la memoria de la GPU
- El tamaño del vector de la representación *Embedded* es un hiperparámetro del modelo que debe ser elegido con cuidado
  - Puede requerir un ajuste iterativo
- El número de capas y el tamaño de cada capa LSTM también son hiperparámetros del modelo que se deben fijar con cuidado
  - Pueden requerir un ajuste iterativo

Caso de estudio: LSTM para IMDB

- Tamaño de cada batch de entrenamiento
  - label: (*batch_size,1*)
  - text: (*batch_size, max_seq_len*) #Padded
- Salida *embedded*: (*batch_size, max_seq_len, embedding_dim)*
- Salida de las capas LSTM (*batch_size, max_seq_len, hidden_dim*)
- Salida de la capa Lineal de salida: (*batch_size, 1*)

Laboratorio

- Revisa el notebook rnns.ipynb
- Realiza el **Ejercicio** al final del notebook