Redes neuronales recurrentes

Minería de datos. Master en Tecnologías de Análisis de Datos Masivos: Big Data

Juan A. Botía

10/03/2021

Introducción

Introducción

- Las redes neuronales convencionales no tienen memoria.
- Cada ejemplo se procesa de manera independiente
- Para procesar secuencias o series temporales, se tiene que mostrar la secuencia entera a la red de una vez.
- ¿No sería mejor?
  - A medida que se leen tokens, procesar la información token a token, mientras se gestiona información sobre tokens anteriores.
- ¿Cómo obtener un modelo de este tipo basándonos en la probabilidad?

Procesamiento de lenguaje natural, modelo probabilista

- Vamos a ver redes recurrentes (RNNs) para predicción de textos.
- Sean los ejemplos de entrenamiento secuencias de T palabras,
  - $x_1, x_2, \ldots, x_T$ en donde la palabra x_t, $1 \leq t \leq T$ es la palabra observada en el instante t.
- Queremos modelar la probabilidad conjunta de la secuencia

$$P(x_1, x_2, \ldots, x_T),$$

- Según este modelo, una vez observadas $x_1, x_2, \ldots, x_{t-1}$, bastaría con preguntar por la probabilidad $P(x_t|x_{t-1}, \ldots, x_1)$, para todas las palabras posibles a predecir.

Es decir la formulación de nuestro problema es

$$P(x_1, x_2, \ldots, x_T) = \prod^{T} P(x_t|x_{t-1}, \ldots, x_1)$$

Un ejemplo

La probabilidad de la sentencia "el deep learning mola"

- bastaría con calcular

$$P(el, deep, learning, mola)$$

mediante

$$P(el)P(deep|el)P(learning|el, deep)P(mola|learning, deep, el).$$

- $P(el)$ se obtiene por la frecuencia de aparición en un texto.
- $P(deep|el)$, mediante la expresión $P(deep|el) = \frac{n(el,deep)}{n(el)}$, siendo $n(w)$ el número de veces que aparece la palabra w.
- El resto es impracticable para cualquier problema serio.

Un ejemplo (y II)

Al sufrir una explosión combinatoria, necesitamos relajar las asunciones que adoptamos a la hora de resolver el problema

- Usamos la condición de Markov

$$P(x_1, x_2, \ldots, x_T) = \prod_{t=1}^{T} P(x_t | x_{t-1}),$$

En un problema de Markov es suficiente usar el estado inmediatamente anterior para hacer inferencia sobre el estado siguiente.

Redes neuronales recurrentes

Redes neuronales recurrentes

- Una red neuronal recurrente (RNN) tiene en cuenta estados anteriores en el tiempo para predecir estados actuales.
- Asume un entorno Markoviano
- Procesa secuencias iterando sobre sus elementos mientras mantiene un estado que contiene información de lo que ha visto hasta el momento.
- Dicho estado se restaura cada vez que se termina de procesar una secuencia para procesar la siguiente, por lo que se sigue considerando una secuencia como una sola entrada a la red.
- Cada muestra se procesa internamente iterando sobre los elementos de la secuencia.

En la figura siguiente podemos verlo.

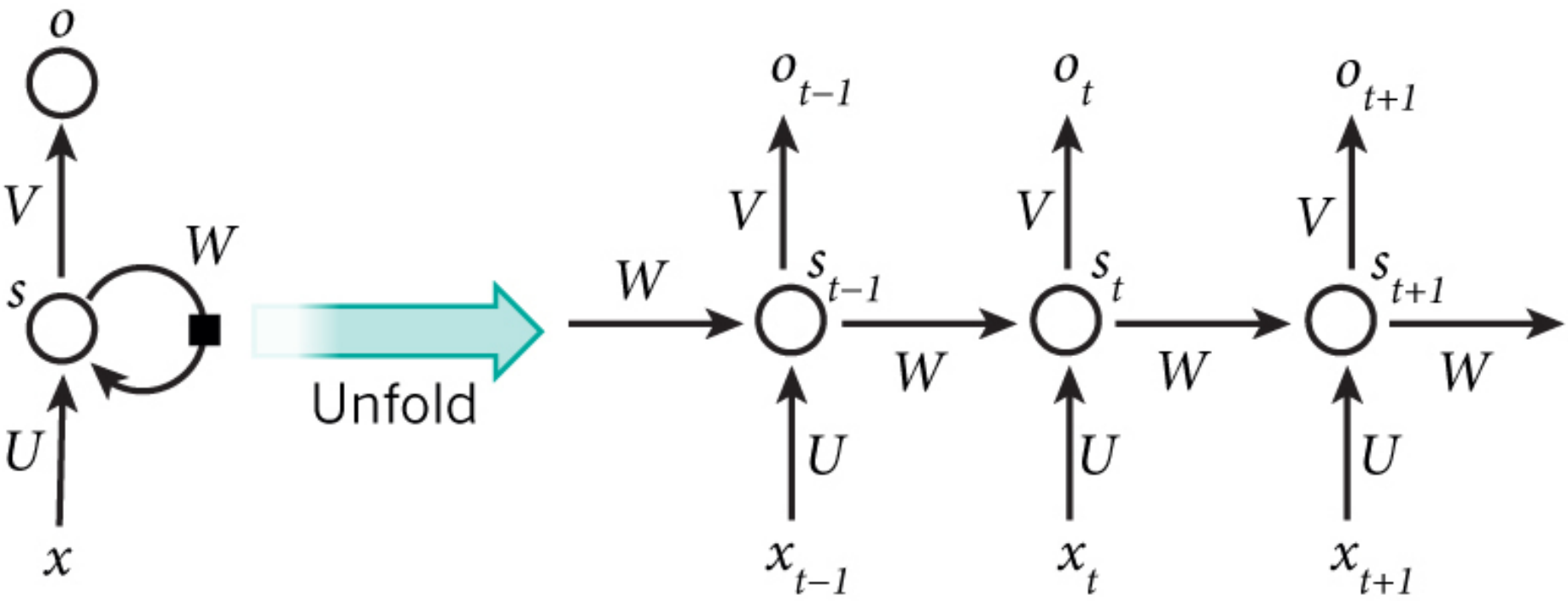


Figure 1: Estructura de una red neuronal recurrente al hacer inferencia

Figura de una RNN, continuación

- x es la entrada, de n palabras de longitud, d es el número de nodos de la red, t instante de tiempo actual y k el número de clases a predecir.
- $U \in R^{n \times d}$ parametriza el estado s, depende de la entrada y de U
- $W \in R^{d \times d}$ parametriza relación entre s_t y el estado anterior s_{t-1}
- $V \in R^{d \times k}$ modula relación entre s_t y la salida o_t en el mismo instante
- $s_t \in R^d$, el estado en t es función de U y W tal que

$$s_t = f(Ux_t + Ws_{t-1} + b_s)$$

  en donde f es una función de activación derivable.
- En lugar de un producto único de matrices, en las RNN tenemos multiplicación recurrente (i.e. cada vez, la expresión es la misma y depende de la misma definición) de matrices hacia atrás.

RNNs, funcionamiento

- U, W y V tienen siempre el mismo tamaño y no cambian conforme se va avanzando en la predicción, dentro de cada ejemplo de entrenamiento.
- La salida de la red, o es una secuencia, ya que cada elemento de un ejemplo concreto x genera una predicción, $o_t \in R^k$, tal que $o_t = f'(Vs_t + b_0)$ siendo $f'()$ otra función de activación (e.g. una softmax).
- En la figura vemos una RNN desenrollada (es realmente simbólico, solo ocurre a nivel de cálculo)

RNNs, funcionamiento (y II)

Supongamos la cadena de entrada a, c, e, g.

- La red lo procesaría como sigue

$$
\begin{aligned}
s_a &= f(Ux_a + b_s) \rightarrow o_a = f'(Vs_a + b_o) \\
s_c &= f(Ux_c + Ws_a + b_s) \rightarrow o_c = f'(Vs_c + b_o) \\
s_e &= f(Ux_e + Ws_c + b_s) \rightarrow o_e = f'(Vs_e + b_o) \\
s_g &= f(Ux_g + Ws_e + b_s) \rightarrow o_g = f'(Vs_g + b_o).
\end{aligned}
$$

- Cada una de estas capas tiene un número de nodos que se le han de especificar cuando se ejecute el modelo
- Deben ser capaces de funcionar para longitudes diferentes de las entradas.
- Necesitamos en el proceso de cada token, la misma función para calcular el estado de la red

Recapitulando

1. El estado s_t puede verse como un nodo con memoria ya que

$$s_t = f(Ux_t + Ws_{t-1} + b_s)$$

2. La salida en t, o_t, se calcula basándonos en lo que sabemos en ese momento
   - Pero podemos asegurar que la salida o_t depende de todas las entradas pasadas, ya que s_t depende de s_{t-1}, que depende de su entrada x_{t-1} y del estado s_{t-2} y así sucesivamente.
3. RNNs necesitean matriz de pesos adicional, en comparación a una MLP de 1 capa, la matriz W, siendo el conjunto de matrices (U, W, V, b) entre todos los pasos.
   - Realizamos la misma tarea para cada token en una secuencia (el número de parámetros se reduce considerablemente).
4. La figura anterior muestra salidas para cada token
   - Puede no ser necesario: para predecir el sentimiento de una frase solo necesitaríamos la salida final de la red (la de la última capa).

El backpropagation through time

Backpropagation through time

Es el algortimo básico para entrenamiento de pesos en las RNN

- Como en todo algoritmo de optimización por gradiente, comenzamos definiendo la función de error,

$$L(h, \hat{y}) = \sum_t L_t(y_t, \hat{y}),$$

en donde L_t es la función de error (loss en inglés), y_t es el valor de la variable a predecir en el instante t y $\hat{y}_t$ es el valor predicho en ese mismo instante.

Backpropagation through time (y II)

Recordemos que en BP clásico

- un ejemplo u se propaga desde los nodos de entrada a los nodos de salida de un MLP.
- la contribución al error de cada nodo de salida i (δ_i), es

$$\delta_i = \hat{y}_i(1 - \hat{y}_i)(y_i - \hat{y}_i)$$

- La contribución de nodos ocultos h al error es

$$\delta_h = \hat{y}_h(1 - \hat{y}_h) \sum_{i \in Outputs} w_{h,i}\delta_i$$

- Los pesos cambian según

$$w_{i,j} \leftarrow w_{i,j} + \Delta w_{i,j},\ \Delta w_{i,j} = \eta \delta_i u_{i,j},$$

- $w_{i,j}$ es el peso de un arco que sale del nodo i-ésimo y llega al nodo j-ésimo,
- u_i, j es la entrada al nodo j-ésimo que llega por el arco i-ésimo
- usamos h para indexar nodos de la capa oculta.

Backpropagation through time (y III)

BPTT ha de tener en cuenta que en el instante t las salidas de cada nodo (y por tanto, la contribución de cada nodo al error), dependen de los valores anteriores del nodo (los valores de s_i, con $1 \leq i \leq t$)

- La derivada de la función de error incluye la expresión no solo en ese instante sino en los anteriores y lo haremos de manerea aditiva.

La figura incluye

- el proceso de inferencia, hacia delante (flechas gruesas de izquierda a derecha),
- hacia dónde cada gradiente de relevancia.

Backpropagation through time (y IV)

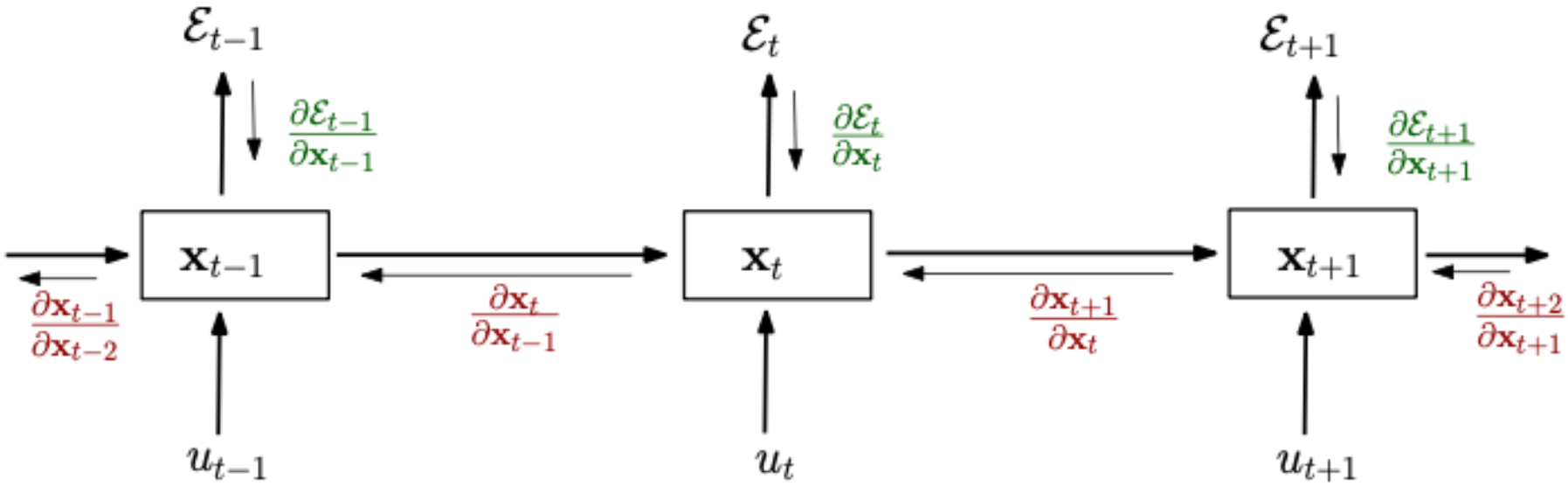


Figure 2. Unrolling recurrent neural networks in time by creating a copy of the model for each time step. We denote by $\mathbf{x}_t$ the hidden state of the network at time t, by $\mathbf{u}_t$ the input of the network at time t and by $\mathcal{E}_t$ the error obtained from the output at time t.

Backpropagation through time (y V)

- La red se desenrolla hacia delante, de tal forma que eventos pasados entran a la siguiente "copia" inmediata de la red.
- El error de la red en el instante actual (ε_t) participa en el gradiente en t,
- Los elementos

$$\frac{\partial \varepsilon_t}{\partial x_t}$$

en verde, son las contribuciones al error del backpropagation convencional, que se aplican en el instante correspondiente.

- Los

$$\frac{\partial x_{t-i}}{\partial x_{t-(i-1)}},$$

viajan hacia atrás (la corrección aplicada a los pesos en t depende de $t+1$, predicciones pasadas contribuyen al error de predicciones futuras)

Backpropagation through time (y VI)

- El error total cometido por el modelo,es la suma de los errores cometidos a cada instante t

$$L(h, \hat{y}) = \sum_t L_t(y_t, \hat{y}).$$

- Nos centramos en la derivada de L con respecto a la matriz W
  - Error se calcula aditivamente a través de todos los instantes de tiempo

$$\frac{\partial L}{\partial W} = \sum_t \frac{\partial L_t}{\partial W}.$$

Backpropagation through time (y VII)

- L_t depende de s_t, que depende de s_{t-1} y de W y así hasta s_1 (depende también de W y de s_0), medianbte la regla de la cadena tenemos

$$\frac{\partial L_t}{\partial W} = \sum_{k=0}^{t} \frac{\partial L_t}{s_t} \frac{\partial s_t}{\partial s_k} \frac{s_k}{\partial W}.$$

- $$\frac{\partial s_t}{\partial s_k} = \prod_{t \geq i \geq k} \frac{\partial s_i}{\partial s_{i-1}} = \prod_{t \geq i \geq k} diag(f'(s_{i-1}))W$$

- $$\frac{\partial s_i}{\partial s_{i-1}}$$

  es una derivada aplicada sobre una matriz de funciones y por eso el resultado tiene como componente a la diagonal, porque las $\frac{\partial s_{i,j}}{\partial s_{i-1,k}} = 0$, cuando $j \neq k$.

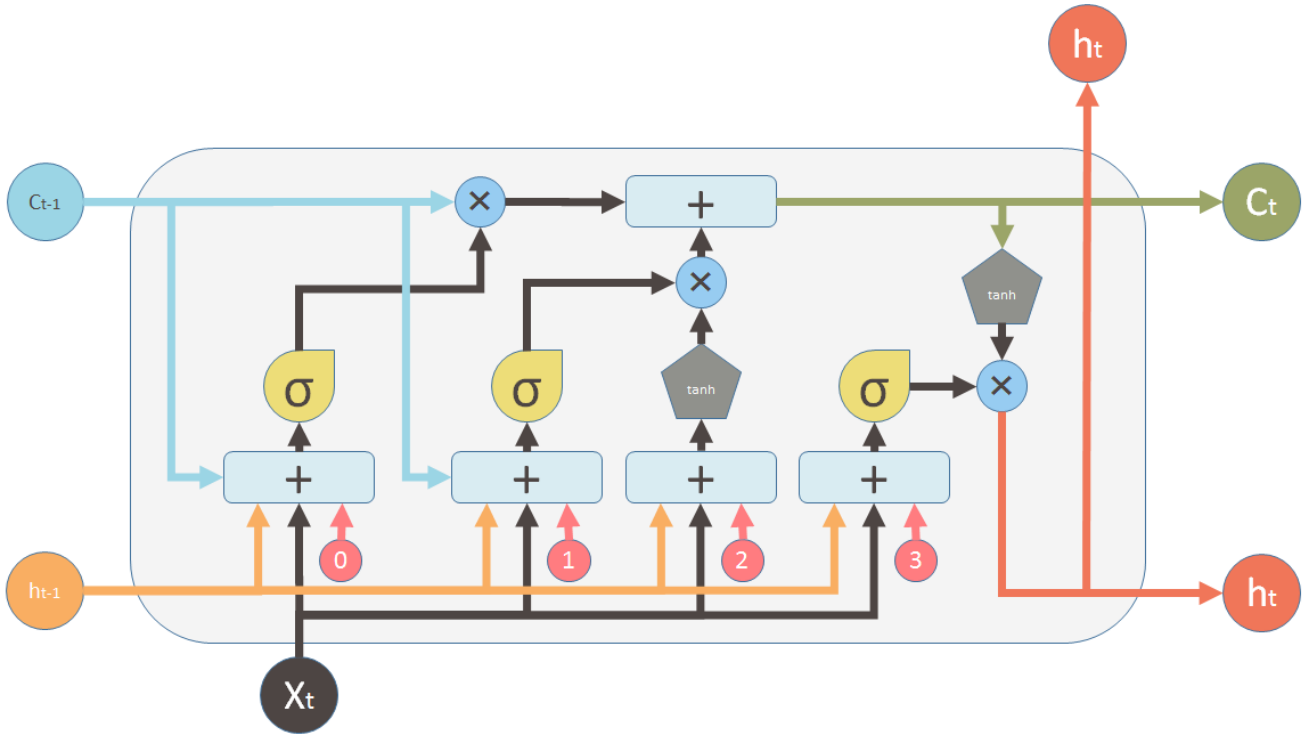
Redes LSTM

Partes de una Red LSTM

Las redes LSTM (Long Short Term Memory) son una clase especial de RNNs capaces de aprender dependencias a largo plazo.

- Introducidas por Hochreiter y Schmidhuber en 1997
- Funcionan bien en una gran variedad de problemas
- Una versión más actual y eficiente son las GRU (Gated Recurrent Units) pero el principio básico es el mismo.
- Un nodo en una ANN LSTM es más complejo que un nodo recurrente clásico.
- Se controla (1) qué cantidad de información se recuerda del estado actual (2) qué cantidad de memoria del estado anterior se va a utilizar en la composición de la memoria del estado actual.
- Además, se aprende durante el entrenamiento.

Esquema del nodo básico LSTM



Inputs:

- X_t Input vector
- C_{t-1} Memory from previous block
- h_{t-1} Output of previous block

outputs:

- C_t Memory from current block
- h_t Output of current block

Nonlinearities:

- σ Sigmoid
- tanh Hyperbolic tangent

Vector operations:

- × Element-wise multiplication
- \+ Element-wise Summation / Concatenation

Bias: 0

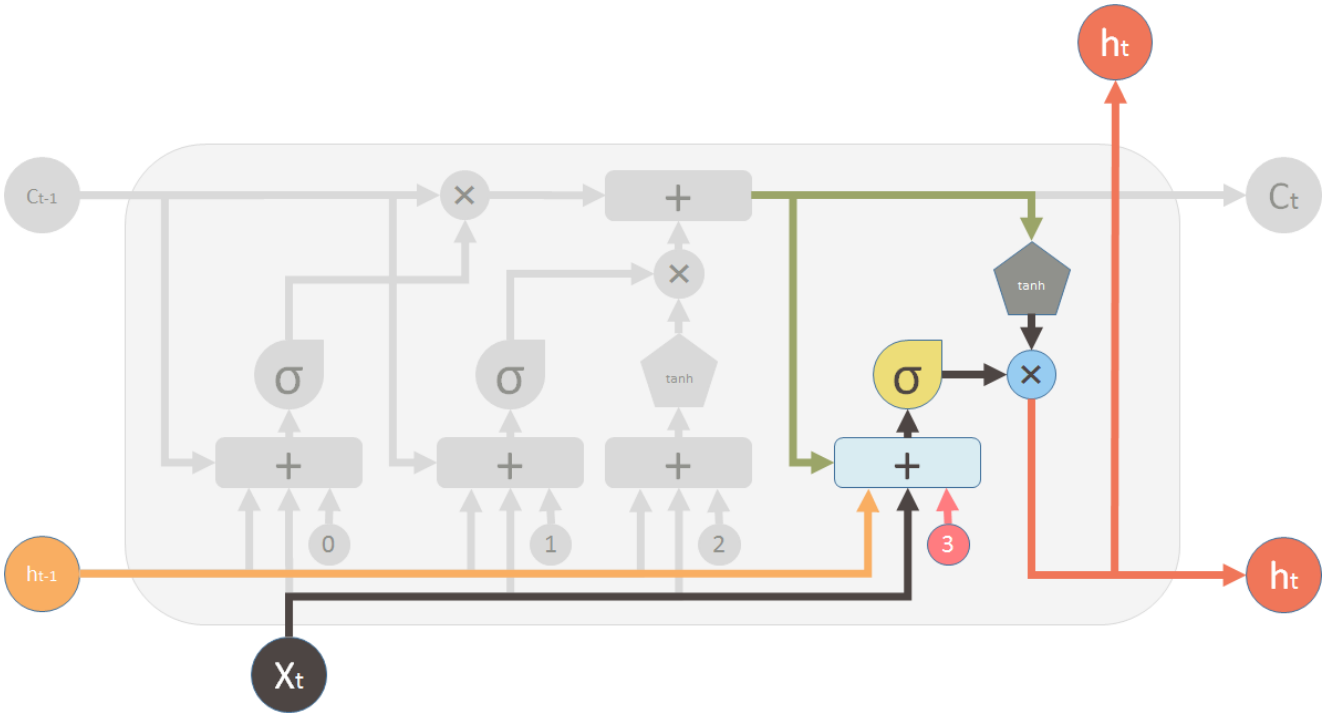
Esquema del nodo básico LSTM (Y II)

El elemento más importante es el estado de la célula, C_t que

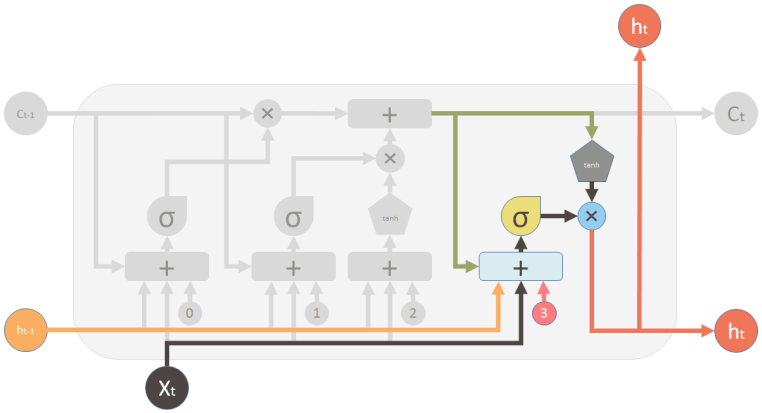
- se refiere a la memoria mantenida por el nodo LSTM
- se crea a partir del flujo que aparece representado de izquierda a derecha en la parte superior del nodo.
- Se alimenta de los cuatro flujos que van de abajo a arriba y se incorporan al flujo principal con simples operaciones vectoriales de producto o suma.
- Cada uno de estos flujos desemboca en una sigmoide (σ), que se refiere a una función sigmoidal que genera valores en $[0, 1]$ o en una tangente hiperbólica que genera valores en $[-1, 1]$.
- Los datos de entrada a un nodo LSTM son la entrada en el instante t correspondiente al ejemplo de entrenamiento actual, el estado oculto h_{t-1}

La puerta de salida o_t

La parte del diagrama anterior que se corresponde con esta es



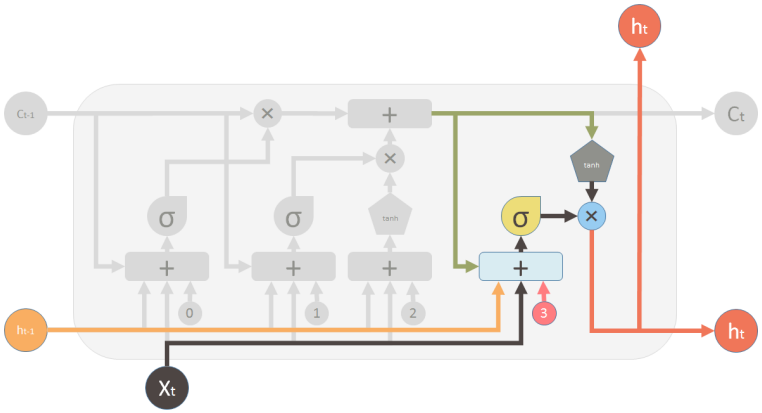
La puerta de salida o_t (y II)



Antes de comenzar, asumimos que

- nos encontramos en $t-1$,
- acabamos de calcular el valor C_{t-1} y
- queremos generar la salida para dicho instante, que es h_{t-1}, para así poder trabajar en el siguiente instante, t.

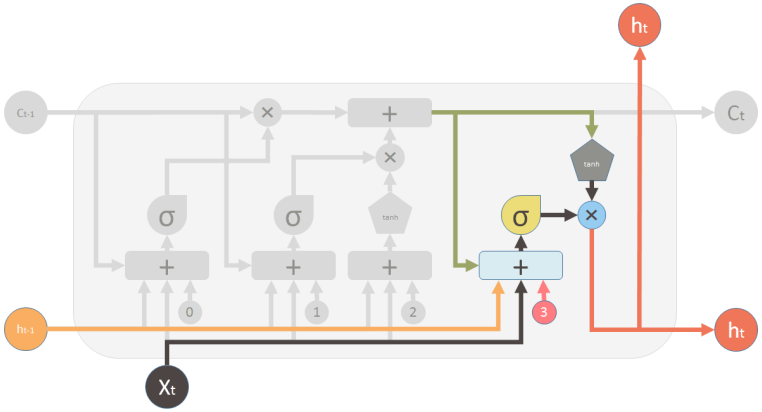
La puerta de salida o_t (y III)



- El nodo naranja (h_t) toma dos entradas:
  - C_t y una sigmoide (que denotamos aquí con o_t) que a su vez se alimenta de la entrada actual, la salida del estado en el instante anterior, h_{t-1} y un sesgo.
  - Y todos estos flujos van a un nodo etiquetado con x que se refiere a un producto vectorial componente a componente. Por tanto,

$$o_{t-1} = \sigma(W_0 h_{t-2} + U_o x_{t-1} + b_0)$$

La puerta de salida o_t (y III)



- Luego $h_{t-1} = o_t \tanh(C_t)$.
- h_t está basada en C_t, pero pasado antes por la sigmoide (modula % de información usado)
- La tangente hiperbólica actúa controlando el crecimiento del gradiente (para que no genere valores excesivamente grandes).
- La LSTM aporta un elemento nuevo a la red: permite entrenar también la parte que decide cómo queremos considerar el estado previo, en la medida en que se modula el % de estado usado.

La puerta de olvido

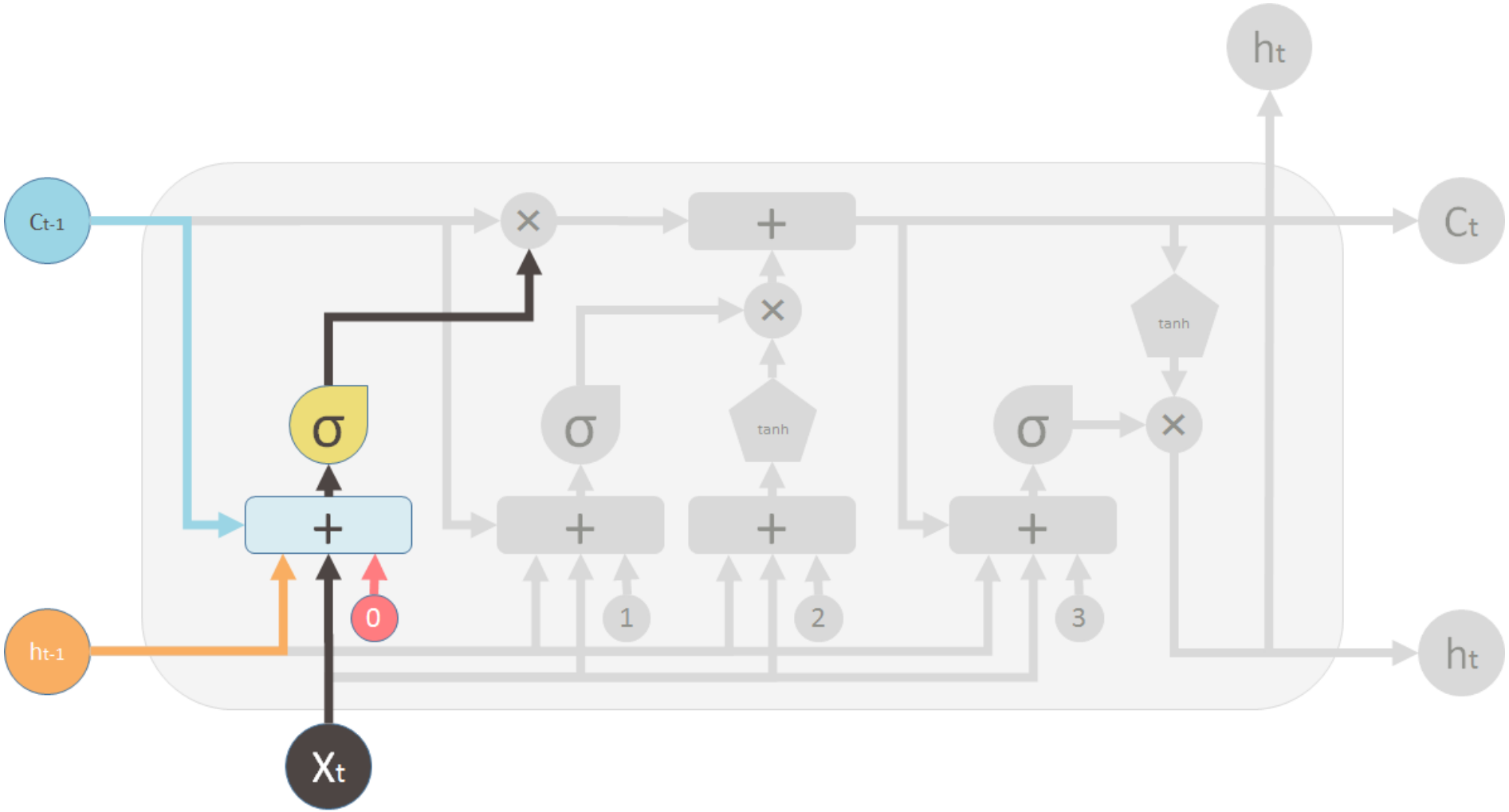


Figure 4: Puerta de olvio de la red LSTM

La puerta de olvido

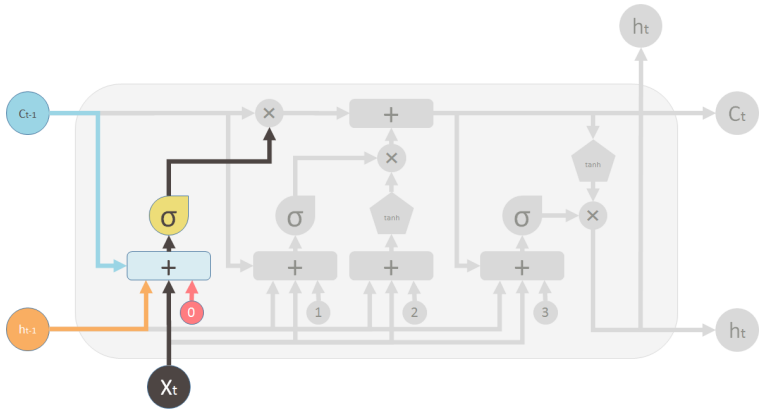


Figure 5: Puerta de olvido de la red LSTM

- Una vez hemos construido la salida h_{t-1} y la hemos pasado a la siguiente celda, el siguiente paso es decidir cómo modular C_{t-1}.
- Se produce mediante sigmoidal, denominado forget layer.
- Recibe h_{t-1} y x_t, y devuelve un vector de números entre 0 y 1 que se multiplica elemento a elemento con el estado C_{t-1}.

En esta puerta de olvido, que denotamos con f_t las ecuaciones son

La puerta de entrada i_t

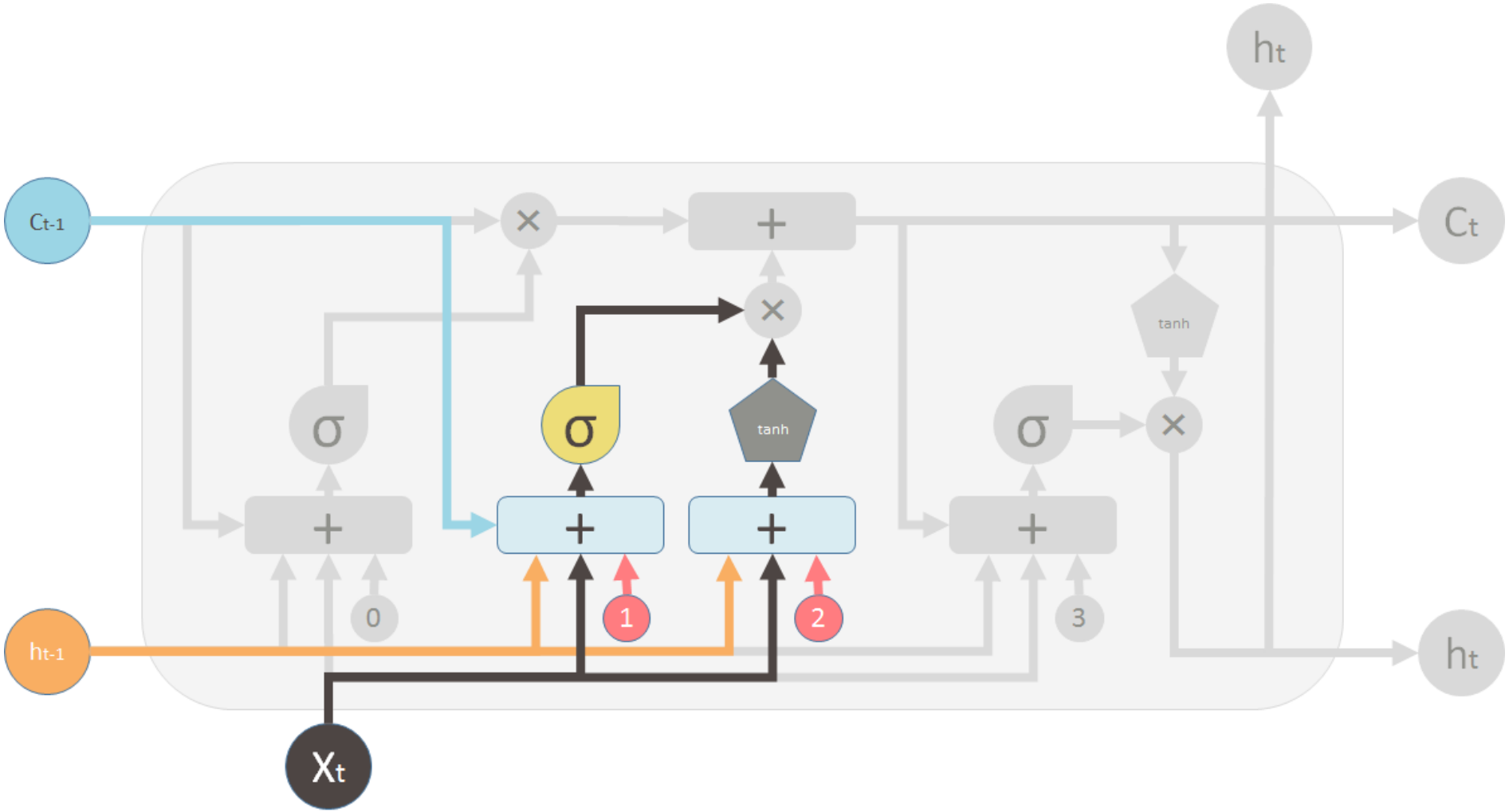


Figure 6: La puerta de entrada i_t

La puerta de entrada i_t (y II)

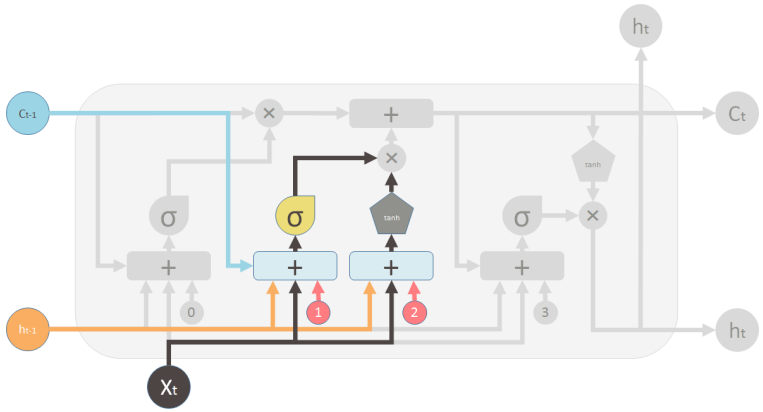


Figure 7: La puerta de entrada i_t

- Ahora necesitamos incorporar a nuestro nodo LSTM el nodo clásico de red recurrente en el que la salida dependerá de la entrada actual y del estado anterior
- también modularemos cuánto de esa salida se genera al exterior, mediante una multiplicación punto con un vector del mismo número de componentes, producidos por una sigmoide.
- Como este estado es un C_t temporal ya que, en realidad, se ha de integrar con C_{t-1} más adelante, lo denotamos con $\widetilde{C}_t$.

La puerta de entrada i_t (y III)

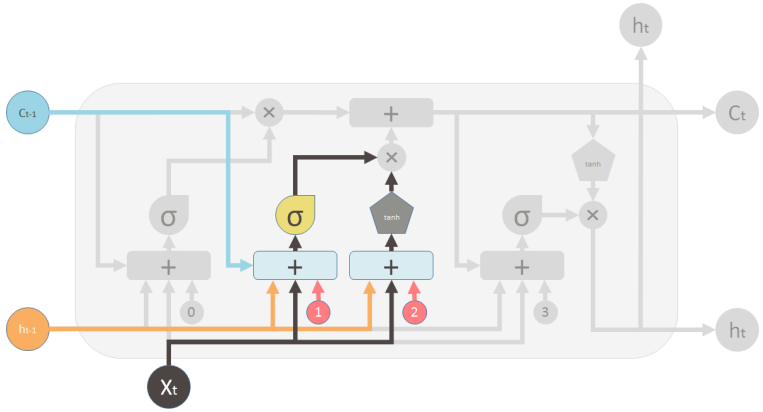


Figure 8: La puerta de entrada i_t

- Una mejora propia de las LSTM: modular la salida con una sigmoide

$$i_t = \sigma(W_i h_{t-1} + U_i x_t + b_i),\ C_t = C_{t-1} \times f_t + i_t \times \widetilde{C}_t$$

* Todo lo que se ha mostrado sobre LSTMs se refiere a una sola capa

- Normalmente se concatenan más capas de LSTM verticalmente para permitir una mayor complejidad a nuestro modelo.

Recapitulando

A modo de resumen,

- las redes LSTM son similares a las RNNs convencionales en el sentido de que tienen un módulo que se repite por cada elemento de la secuencia y cada módulo comparte los mismos pesos (parámetros).
- Sin embargolas redes LSTM están controladas por puertas que controlan qué información se debe pasar u olvidar al siguiente estado, mientras que las RNN pasaban toda la información del estado previo al actual.

Bidireccionalidad en redes recurrentes

Capas bidireccionales

Supongamos que queremos predecir la palabra que va en un hueco intermedio de una frase. Por ejemplo, supongamos que tenemos las frases:

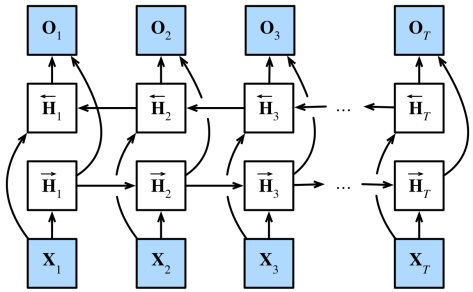
1. Tengo
2. Tengo hambre.
3. Tengo hambre, me podría comer cinco hamburguesas.

Podemos rellenar los huecos con palabras como “sueño”, “poca” o “mucha”.

- No es suficiente con las palabras previas, es necesario mirar a las palabras posteriores
- Trabajar de izquierda a derecha, y de derecha a izquierda, i.e. necesitamos redes bidireccionales

Capas bidireccionales (y II)

Necesitamos una red que propague errores hacia atrás y hacia delante.



Capas bidireccionales (y III)

- No solo se ha de tener en cuenta el estado s_t
- Necesitamos un nuevo s'_t, que vaya en sentido contrario.
- s_t se asocia con $\overrightarrow{H}$ y s'_t con $\overleftarrow{H}$.
- Cada estado se procesará de forma diferente tal que

$$\overrightarrow{H}_t = f(X_t W^{(f)} + \overrightarrow{H}_{t-1}\, U^{(f)} + b^{(f)})$$
$$\overleftarrow{H}_t = f(X_t W^{(b)} + \overleftarrow{H}_{t-1}\, U^{(b)} + b^{(b)})$$

para los casos forward (f) y backwards (b).

- La inferencia final concatena $\overrightarrow{H}_t$ y $\overleftarrow{H}_t$ en un solo H_t para hacer la inferencia como siempre, según $o_t = o(VH_t + b_o)$.

Referencias

Referencias

En este tutorial se estudian a fondo las redes recurrentes. La figura 1 la hemos tomado de ahí.

El material para el BPTT se ha extraido del paper https://arxiv.org/abs/1211.5063

Las figuras de los diferentes nodos LSTM son de https://medium.com/mlreview/understanding-lstm-and-its-diagrams-37e2f46f1714

Para seguir profundizando, mirar los libros https://d2l.ai/, https://www.deeplearningbook.org/ y https://web.stanford.edu/~jurafsky/slp3/

Rafael Jordá, antiguo alumno de este master, ha contribuido con material para el LSTM de su TFM.