Redes neuronales recurrentes

Víctor Mijangos

Facultad de Ciencias

Lingüística computacional

Datos secuenciales

Si bien las redes neuronales pre-alimentadas son capaces de aproximar funciones de interés, muchos problemas nos enfrentan a la predicción de cadenas, a partir de cadenas.

Por ejemplo, la traducción automática requiere transformar una secuencia de palabras en otra secuencia en otro idioma:

$$X^{(1)}, X^{(2)}, ..., X^{(n)} \rightarrow Y^{(1)}, ..., Y^{(m)}$$

Datos secuenciales

- Nos enfrentamos con procesos estocásticos, es decir una secuencia de variables aleatorias $\{X^{(t)}\}_t$, donde existe dependencia de los elementos anteriores.
- Una red neuronal pre-alimentada para cada variable requiere de un esfuerzo exhaustivo, además de que no preserva la información en estados anteriores.

Redes neuronales recurrentes

Propuesta: Compartir parámetros a partir de diferentes partes (tiempos) del modelo.
Ventaja: Se pueden detectar elemenos iguales en posiciones diferentes en una secuencia.

Las **Redes Neuronales Recurrentes** o RNNs [1] es una familia de redes neuronales orientadas a procesar datos secuenciales. Cada elemento de salida es una función de las salidas anteriores.

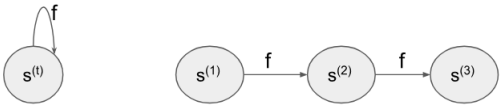
Despliegue de redes

Tenemos funciones que tomen en cuenta los estados anteriores para definir los estados actuales:

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

Desplegar este tipo de funciones consiste en aplicar la función las veces necesarias:

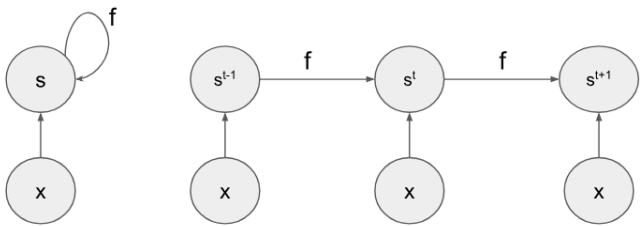
$$s^{(5)} = f(f(f(s^{(1)}; \theta); \theta); \theta)$$



Despliegue de redes

Podemos tomar en cuenta una entrada $x^{(t)}$ en este tipo de funciones:

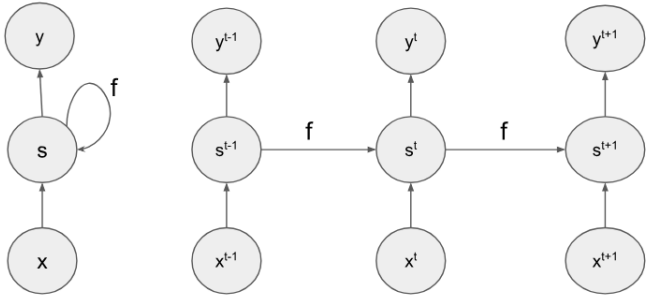
$$s^{(t)} = f(s^{(t-1)}, x^{(t)}; \theta) \tag{2}$$



Redes neuronales recurrentes

Más aún, se pueden definir funciones que, además de una entrada, emitan una salida $y^{(t)}$:

$$\begin{pmatrix} s^{(t)} \\ y^{(t)} \end{pmatrix} = f(s^{(t-1)}, x^{(t)}; \theta) \tag{3}$$



Redes neuronales recurrentes

Una RNN típica mapea de una secuencia a otra: $x^{(1)}, x^{(2)}, ..., x^{(T)} \mapsto y^{(1)}, y^{(2)}, ..., y^{(T)}$.

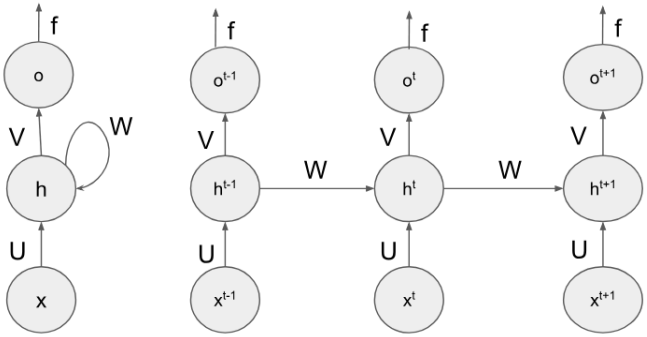
Definimos este tipo de RNNs a partir de los siguientes elementos:

- $a^{(t)} = Wh^{(t-1)} + Ux^{(t)} + b$
- $h^{(t)} = g(a^{(t)})$
- $o^{(t)} = Vh^{(t)} + c$
- $\hat{y}^{(t)} = f(o^{(t)})$

Generalmente, las $f \equiv Softmax$ y $g \equiv \sigma$ ó $g \equiv \tanh$.

Redes neuronales recurrentes

La RNN comparte los parámetros *U*, *W* y *V* a través de los diferentes estados de la secuencia.



Entrenamiento

El entrenamiento de una RNN es similar al de otra red neuronal. La función de pérdida se define como:

$$R(\hat{y}, y) = \sum_{\hat{y}} \sum_{t=1}^{T} R(\hat{y}^{(t)}, y^{(t)}) = -\sum_{\hat{y}} \sum_{t=1}^{T} y^{(t)} \ln p(y^{(t)} | x^{(1)}, ..., x^{(T)}) \tag{4}$$

La retro-propagación es similar a la de otras redes neuronales. En este caso, se toma en cuenta el estado (el tiempo).

Retro-propagación sobre el tiempo

Para retro-propagar el error en una RNN, tomamos los parámetros U, V, W y los bias b, c.
Entonces:

- Se comienza la recursión después de calcular R:

$$\frac{\partial R(\hat{y}, y)}{\partial R(\hat{y}^{(t)}, y^{(t)})} = \frac{\partial \sum_t R(\hat{y}^{(t)}, y^{(t)})}{\partial R(\hat{y}^{(t)}, y^{(t)})} = 1$$

- Para la capa de salida, observamos que:

$$\begin{aligned}\frac{\partial R(\hat{y}^{(t)}, y^{(t)})}{\partial V_{ij}} &= \frac{\partial R}{\partial f_i^{(t)}} \frac{\partial f_i^{(t)}}{\partial o_i^{(t)}} \frac{\partial o_i^{(t)}}{\partial V_{ij}} \\ &= (f_i^{(t)} - y_i^{(t)}) h_j^{(t)} \\ &= \mathbf{d_{out}}(\mathbf{i}; \mathbf{t}) h_j^{(t)}\end{aligned}$$

Retro-propagación sobre el tiempo

En el caso de la capa oculta, tenemos dos matrices sobre las que derivaremos U y W. Para U en un tiempo t tenemos que:

$$\begin{aligned}\frac{\partial R(\hat{y}^{(t)}, y^{(t)})}{\partial U_{ij}} &= \frac{\partial R}{\partial f_i^{(t)}} \frac{\partial f_i^{(t)}}{\partial o_i^{(t)}} \frac{\partial o_i^{(t)}}{\partial h_i^{(t)}} \frac{\partial h_i^{(t)}}{\partial a_i^{(t)}} \frac{a_i^{(t)}}{\partial U_{ij}} \\ &= [\sum_q V_{qi} d_{out}(q;t)] \frac{\partial h_i^{(t)}}{\partial a_i^{(t)}} x_j^{(t)} \\ &= \mathbf{d_h}(\mathbf{i};\mathbf{t}) x_j^{(t)}\end{aligned}$$

Tanto la derivación en para esta matriz como en la capa de salida son similares a la de una red FeedForward, con el añadido del **tiempo**.

Retro-propagación sobre el tiempo

Finalmente, para la matriz W, que considera los estados anteriores, tenemos que la derivación es:

$$\begin{aligned}\frac{\partial R(\hat{y}^{(t)}, y^{(t)})}{\partial W_{ij}} &= \frac{\partial R}{\partial f_i^{(t)}} \frac{\partial f_i^{(t)}}{\partial o_i^{(t)}} \frac{\partial o_i^{(t)}}{\partial h_i^{(t)}} \frac{\partial h_i^{(t)}}{\partial a_i^{(t)}} \frac{a_i^{(t)}}{\partial W_{ij}} \\ &= [\sum_q V_{qi} d_{out}(q;t)] \frac{\partial h_i^{(t)}}{\partial a_i^{(t)}} h_j^{(t-1)} \\ &= \mathbf{d_h}(\mathbf{i};\mathbf{t}) h_j^{(t-1)}\end{aligned}$$

Recuérdese que $h^{(0)} = \overline{0}$, es el vector de ceros.

Retro-propagación sobre el tiempo

Finalmente, para actualizar los pesos usaremos el gradiente descendiente. En cada tiempo t tendremos:

$$V_{ij} \leftarrow V_{ij} - \eta d_{out}(i;t) h_j^{(t)}$$

$$U_{ij} \leftarrow U_{ij} - \eta d_h(i;t) x_j^{(t)}$$

$$W_{ij} \leftarrow W_{ij} - \eta d_h(i;t) h_j^{(t-1)}$$

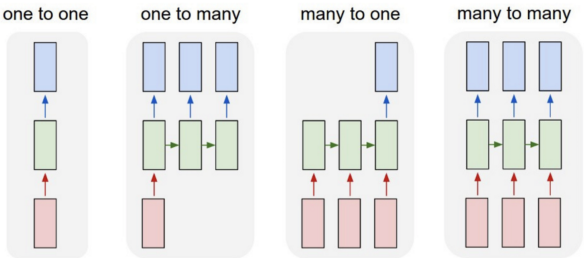
Si la red recurrente tiene capas intermedias entre la entrada y la recurrencia o entre la recurrencia y la entrada estas se actualizarán con el método de backpropagation a través del tiempo.

Otros tipos de RNNs

El número de elementos de entrada y de salida no necesariamente debe coincider. Podemos pensar una RNN como un mapeo:

$$x^{(1)}, ..., x^{(T_x)} \mapsto y^{(1)}, ..., y^{(T_y)}$$

donde $T_x \neq T_y$. Tenemos varias formatos de RNN según la longitud de las cadenas de entrada y de salida:



Otros tipos de RNNs

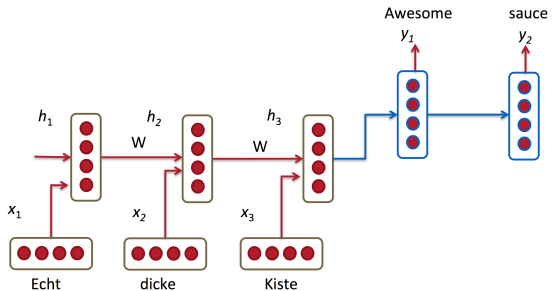
- El modelo one-to-one corresponde a una red neuronal pre-alimentada común.
- En el modelo de one-to-many, se puede tomar la salida del estado anterior como entrada en el estado actual:

$$h^{(t)} = g(Wh^{(t-1)} + Uo^{(t-1)} + b)$$

Otros tipos de RNNs

En los modelos many-to-many también puede pasar que la entrada y la salida sean de diferente longitud.

En este caso, la primera parte se conoce como “encoder” y la segunda como “decoder”:



RNNs bidireccionales

Las RNN que hemos definido recuperan sólo los estados anteriores, pero no son capaces de incorporar información de los estados siguientes. Es decir, son celdas de avance dadas como:

$$\overrightarrow{h}^{(t)} = g(V\overrightarrow{h}^{(t-1)} + Ux^{(t)} + b)$$

Si recorriéramos la cadena en sentido contrario, tendríamos información sobre los estados subsiguientes. En este sentido, también se pueden incorporar celdas de retroceso, es decir, que toman información de los estados siguientes:

$$\overleftarrow{h}^{(t)} = g(\overleftarrow{V}\overleftarrow{h}^{(t+1)} + \overleftarrow{U}x^{(t)} + \overleftarrow{b})$$

Los parámetros $\overleftarrow{V}$, $\overleftarrow{U}$ y $\overleftarrow{b}$ difieren de los de la celda de avance.

RNNs bidireccionales

En una RNN podemos incorporar tanto celdas de avance como de retroceso, estas celdas son independientes. Por lo que su entrenamiento se hará de forma similar a las RNN comunes, excepto por la capa de salida, en donde se incorpora la información sobre ambas celdas:

$$\hat{y}^{(t)} = f(W \overrightarrow{h}^{(t)} + W' \overleftarrow{h}^{(t)} + c)$$

RNNs bidireccionales

Una RNN bidireccional es una arquitectura de RNN, la cual, dada una secuencia de entrada $x^{(1)}, ..., x^{(T_x)}$, determina:

1. **Celda de avance:** Recorre la cadena de inicio a fin. Se define como:

$$\overrightarrow{h}^{(t)} = g(V\overrightarrow{h}^{(t-1)} + Ux^{(t)} + b)$$

2. **Celda de retroceso:** Recorre la cadena de fin a inicio. Se define como[1]:

$$\overleftarrow{h}^{(t)} = g(\overleftarrow{V}\overleftarrow{h}^{(t+1)} + \overleftarrow{U}x^{(t)} + \overleftarrow{b})$$

3. **Variables de salida:** Toma las celdas de avance y retroceso para decidir una salida:

$$\phi(x^{(t)}) = f(W\overrightarrow{h}^{(t)} + W'\overleftarrow{h}^{(t)} + c)$$

[1] Es común que las funciones de activación coincidan en ambas celdas. Por tanto, hemos denotado a ambas g. Sin embargo, es posible también que estas funciones de activación varíen entre la celda de avance y de retroceso

RNNs bidireccionales

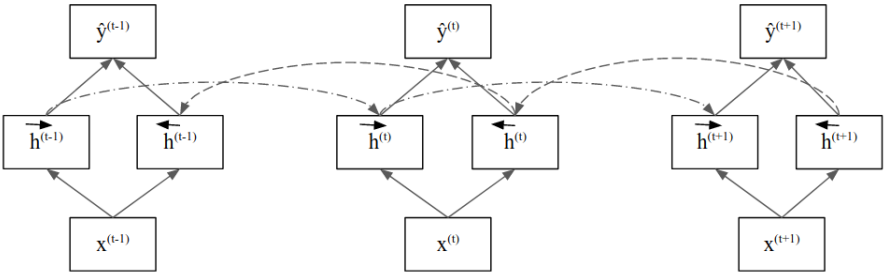


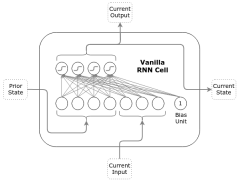
Figura: Versión gráfica de una RNN bidireccional

Vanilla RNN y LSTM

Un problema común en las RNNs son las dependencias a largo pazo y el desvanecimiento de gradiente. Para esto, se proponen diferentes métodos:

- Uso de celdas vanilla. La celda vanilla es la celda típica de RNN definida por

$$h^{(t)} = g(Wh^{(t-1)} + Ux^{(t)} + b)$$



- Lonsg-Short Term Memories (LSTMs) [2]. Buscan asegurar la integridad del mensaje original, escribiéndolo.

LSTMs

Principio fundamental de LSTMs: Para asegurar la integridad del mensaje origina, lo escribimos.
Escribir se entiende como un *cambio incremental aditivo*, que no se modifica por interferencia externa. En términos matemáticos, dada un estado $h^{(t)}$:

$$h^{(t)} = h^{(t-1)} + \Delta h^{(t)} \tag{5}$$

LSTMs

Sin embargo, la escritura puede ser descontrolada o aleatoria. Para solucionar esto, necesitamos **selectividad**:

- ¿Qué escribimos? Escribir la información relevante y no sobre-escribir.
- ¿Qué leemos? Lo más informativo
- ¿Qué olvidamos? Lo poco informativo

Para realizar estos criterios de selectividad, utilizaremos **puertas**:

- Escritura $i^{(t)}$
- Lectura $o^{(t)}$
- Olvido $f^{(t)}$

LSTM

En este caso, cada puerta se comporta de la siguiente forma:

$$
\begin{aligned}
i^{(t)} &= \sigma\left(V^i h^{(t-1)} + U^i x^{(t)} + b^i\right) \\
o^{(t)} &= \sigma\left(V^o h^{(t-1)} + U^o x^{(t)} + b^o\right) \\
f^{(t)} &= \sigma\left(V^f h^{(t-1)} + U^f x^{(t)} + b^f\right)
\end{aligned}
$$

Y proponemos una variable de lectura:

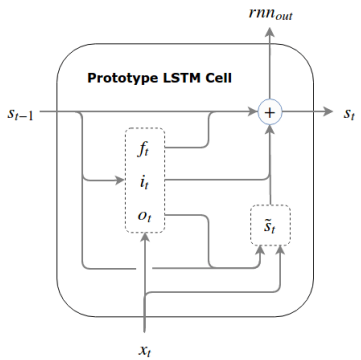
$$
\hat{h}^{(t)} = \psi\left(V\left(o^{(t)} \odot h^{(t-1)}\right) + U x^{(t)} + b\right) \tag{6}
$$

Así, la lectura se da a partir de la escritura, de donde $\Delta h^{(t)} = i_t \odot \hat{h}^{(t)}$. Integrando el olvido, tenemos:

$$
h^{(t)} = f_t \odot h^{(t-1)} + i_t \odot \hat{h}^{(t)} \tag{7}
$$

LSTM

A la integración de esta celda $h^{(t)}$ se le conoce como Prototipo de LSTM:



LSTM

Sin embargo, $\hat{h}^{(t)}$ no es la mejor opción, pues puede saturar.

Se propone escribir después de leer. Para esto se usa una sombra $\hat{c}^{(t)}$ a partir de una celda $c^{(t)}$.

En una LSTM se reciben dos elementos del estado previo: $h^{(t-1)}$ y $c^{(t-1)}$.

En este caso, tenemos que:

$$h^{(t-1)} = o^{(t-1)} \odot \tanh(c^{(t-1)})$$

LSTM

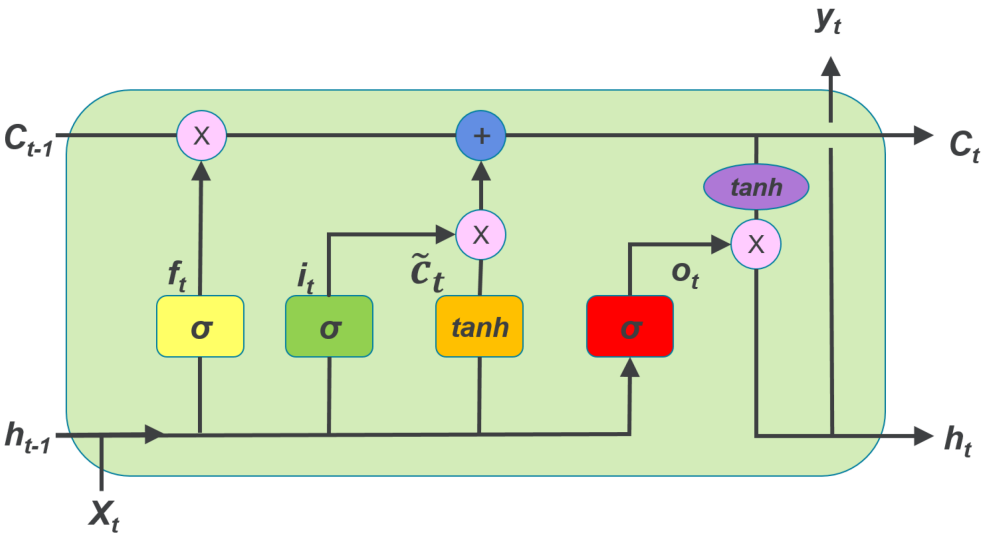
Los valores $i^{(t)}$, $o^{(t)}$ y $f^{(t)}$ permanecen igual (exceptuando el valor de $h^{(t-1)}$. Definimos:

$$\hat{c}^{(t)} = \tanh(Vh^{(t-1)} + Ux^{(t)} + b)$$
$$c^{(t)} = f^{(t)} \odot c^{(t-1)} + i^{(t)} \odot \hat{c}^{(t)}$$

Finalmente:

$$h^{(t)} = o^{(t)} \odot \tanh(c^{(t)}) \tag{8}$$

LSTM



Derivación en celdas LSTM

Para derivar los pesos en las celdas de LSTM hace falta observar cómo se comportan éstas y cómo están jerarquizadas. Dada una función de riesgo, procedemos a derivar desde la capa de salida hasta la capa de entrada. Como nuestra capa de salida es de la forma:

$$\phi(x^{(t)}) = f(Wh^{(t)} + d)$$

Sea, entonces $a^{(out)} = Wh^{(t)} + d$ la preactivación en la capa de salida (asumimos el tiempo):

$$\frac{\partial R}{\partial W_{i,j}} = \frac{\partial R}{\partial \phi_j} \frac{\partial \phi_j}{\partial a_j^{out}} \frac{\partial a_j^{out}}{\partial W_{i,j}}$$

definimos una variable para la capa de salida (estas variables dependen del estado en que se está):

$$d_{out}(j;t) = \frac{\partial R}{\partial \phi_j} \frac{\partial \phi_j}{\partial a_j^{out}}$$

Derivación de o

Para la matriz V^o, tenemos:

$$\frac{\partial R}{\partial V^o_{i,j}} = \frac{\partial R}{\partial h^{(t)}_j} \frac{\partial h^{(t)}_j}{\partial o^{(t)}_j} \frac{\partial o^{(t)}_j}{\partial a^o_j} \frac{\partial a^o_j}{\partial V^o_{i,j}}$$

Para la matriz U^o, la derivación es similar, excepto por la última derivada parcial:

$$\frac{\partial R}{\partial U^o_{i,j}} = \frac{\partial R}{\partial h^{(t)}_j} \frac{\partial h^{(t)}_j}{\partial o^{(t)}_j} \frac{\partial o^{(t)}_j}{\partial a^o_j} \frac{\partial a^o_j}{\partial U^o_{i,j}}$$

Podemos definir la siguiente variable:

$$\begin{aligned} d_o(j;t) &= \frac{\partial R}{\partial h^{(t)}_j} \frac{\partial h^{(t)}_j}{\partial o^{(t)}_j} \frac{\partial o^{(t)}_j}{\partial a^o_j} \\ &= [\sum W_{j,q} d_{out}(q;t)] \tanh(c^{(t)}_j) o^{(t)}_j (1 - o^{(t)}_j) \end{aligned}$$

Derivación de LSTM

Para las puertas de lectura i, olvido f y de candidato $\hat{c}$, tenemos:

$$\frac{\partial R}{\partial V^i_{i,j}} = \frac{\partial R}{\partial h^{(t)}_j} \frac{\partial h^{(t)}_j}{\partial c^{(t)}_j} \frac{\partial c^{(t)}_j}{\partial i^{(t)}_j} \frac{\partial i^{(t)}_j}{\partial a^i_j} \frac{\partial a^i_j}{\partial V^i_{i,j}}$$

$$\frac{\partial R}{\partial V^f_{i,j}} = \frac{\partial R}{\partial h^{(t)}_j} \frac{\partial h^{(t)}_j}{\partial c^{(t)}_j} \frac{\partial c^{(t)}_j}{\partial f^{(t)}_j} \frac{\partial f^{(t)}_j}{\partial a^i_j} \frac{\partial a^i_j}{\partial V^f_{i,j}}$$

$$\frac{\partial R}{\partial V_{i,j}} = \frac{\partial R}{\partial h^{(t)}_j} \frac{\partial h^{(t)}_j}{\partial c^{(t)}_j} \frac{\partial c^{(t)}_j}{\partial \hat{c}^{(t)}_j} \frac{\partial \hat{c}^{(t)}_j}{\partial a^c_j} \frac{\partial a^c_j}{\partial V_{i,j}}$$

Variable de estado

Todas estas derivadas se repite la derivación $\frac{\partial R}{\partial h_j^{(t)}} \frac{\partial h_j^{(t)}}{\partial c_j^{(t)}}$. Por tanto, podemos guardar la información de esta derivación en una variable aparte, que llamaremos d_t y definiremos como:

$$\begin{aligned} d_t(j;t) &= \frac{\partial R}{\partial h_j^{(t)}} \frac{\partial h_j^{(t)}}{\partial c_j^{(t)}} \\ &= [\sum_q W_{j,q} d_{out}(q;t)] o_j^{(t)} (1 - \tanh^2(c_j^{(t)})) + d_t(j;t-1) f^{(t-1)} \end{aligned}$$

Variables backpropagation en LSTM

Capa de salida:

$$d_{out}(j;t) = \frac{\partial R}{\partial \phi_j}\frac{\partial \phi_j}{\partial a_j^{out}}$$

Celda LSTM:

- Para la puerta de lectura:

$$d_o(j;t) = [\sum_q W_{j,q} d_{out}(q;t)] \tanh(c_j^{(t)}) o_j^{(t)}(1-o_j^{(t)})$$

- Para la puerta de escritura:

$$d_i(j;t) = d_t(j;t)\hat{c}_j^{(t)}[i_j^{(t)}(1-i_j^{(t)})]$$

- Para la puerta de olvido:

$$d_i(j;t) = d_t(j;t)c_j^{(t-1)}[f_j^{(t)}(1-f_j^{(t)})]$$

- Para el candidato a escritura:

$$d_c(j;t) = d_t(j;t)i_j^{(t-1)}[1-(\hat{c}_j^{(t)})^2]$$

GD en LSTM

- Para los parámetros de lectura:

$$V^o_{i,j} \leftarrow V^o_{i,j} - \eta d_o(j;t) h_i^{(t-1)}$$

$$U^o_{i,j} \leftarrow U^o_{i,j} - \eta d_o(j;t) x_i^{(t)}$$

- Para los parámetros de escritura:

$$V^i_{i,j} \leftarrow V^i_{i,j} - \eta d_i(j;t) h_i^{(t-1)}$$

$$U^i_{i,j} \leftarrow U^i_{i,j} - \eta d_i(j;t) x_i^{(t)}$$

- Para los parámetros de olvido:

$$V^f_{i,j} \leftarrow V^f_{i,j} - \eta d_f(j;t) h_i^{(t-1)}$$

$$U^f_{i,j} \leftarrow U^f_{i,j} - \eta d_f(j;t) x_i^{(t)}$$

- Finalmente, para los parámetros del candidato:

$$V_{i,j} \leftarrow V_{i,j} - \eta d_c(j;t) h_i^{(t-1)}$$

$$U_{i,j} \leftarrow U_{i,j} - \eta d_c(j;t) x_i^{(t)}$$

Aplicación de RNNs: ELMo

[3] proponen una forma novedosa de representar las palabras en espacios vectoriales.
Su idea radica en dos puntos:

1. Representar las características complejas del uso de las palabras.
2. Determinar cómo estas características varían a través de los contextos.

Su propuesta es llamada **Embeddings form Language Models** (ELMo).

ELMo

ELMo utiliza una RNN biLSTM (bidireccional con celdas LSTM) para determinar un modelo del lenguaje. La función de riesgo está dada por:

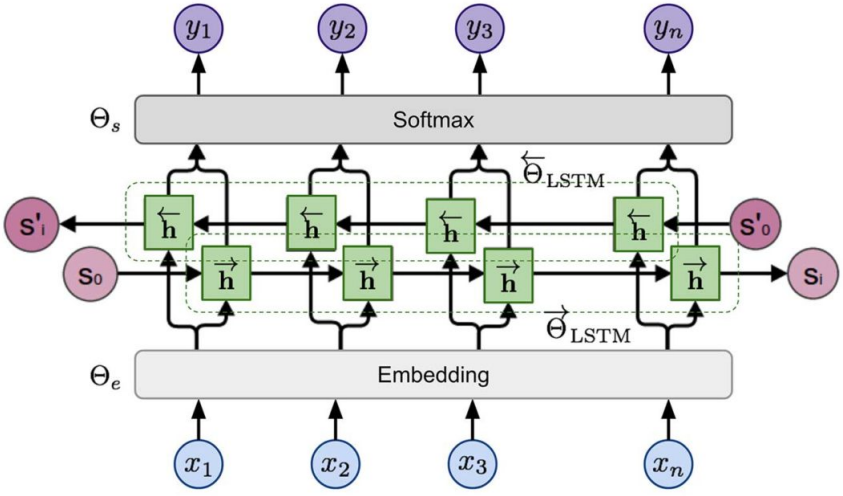
$$\arg\max_{\Theta} \sum_{t=1}^{T} [\log p(w_t|w_1, ..., w_{t-1}) + \log p(w_t|w_{t+1}, ..., w_T)]$$

Que puede reducirse a :

$$\arg\min_{\Theta} - \sum_{t=1}^{T} \log p(w_t|w_1, ..., w_{t-1}, w_{t+1}, ..., w_T)$$

Tal que $\Theta = \{\Theta_x, \Theta_s, \overrightarrow{\Theta}_{LSTM}, \overleftarrow{\Theta}_{LSTM}\}$ son los parámetros de la RNN.

ELMo


y_1
y_2
y_3
y_n
Θ_s
Softmax
$\overleftarrow{\Theta}_{\text{LSTM}}$
$\mathbf{s'}_i$
$\overleftarrow{\mathbf{h}}$
$\mathbf{s'}_0$
$\mathbf{s}_0$
$\overrightarrow{\mathbf{h}}$
$\mathbf{s}_i$
$\overrightarrow{\Theta}_{\text{LSTM}}$
Θ_e
Embedding
x_1
x_2
x_3
x_n

ELMo

Se pueden utilizar K capas biLSTM, de tal forma para cada palabra w_t en el estado t se determinan las representaciones:

$$R_t = \{h^{(t,k)} : k = 0, 1, ..., K\}$$

Tal que ($C(t)$ es la palabra en la capa de embedding):

$$h^{(t,k)} = \begin{cases} C(t) & \text{si } k = 0 \\ [\overrightarrow{h}^{(t,k)}; \overleftarrow{h}^{(t,k)}] & \text{si } k \geq 1 \end{cases}$$

Representaciones ELMo

La representación contextualizada de una palabra w_t está dada por una combinación lineal de la forma:

$$ELMo(w_t) = E(R_t; \Theta) = \gamma \sum_{k=0}^{K} s_k h^{(t,k)}$$

Donde s_k son pesos normalizados a partir de una función Softmax y γ es un parámetro que escala los vectores obtenidos.

References

Jeffrey L Elman.
Finding structure in time.
Cognitive science, 14(2):179–211, 1990.

Sepp Hochreiter and Jürgen Schmidhuber.
Long short-term memory.
Neural computation, 9(8):1735–1780, 1997.

Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer.
Deep contextualized word representations.
arXiv preprint arXiv:1802.05365, 2018.

The End