Redes neuronales recurrentes

Mtro. Víctor Mijangos

Facultad de Ingeniería

13-17 de enero, 2019

Datos secuenciales

Si bien las redes neuronales pre-alimentadas son capaces de aproximar funciones de interés, muchos probleas 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 **Reds 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.

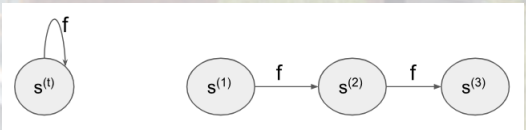
Despliegue de redes

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:

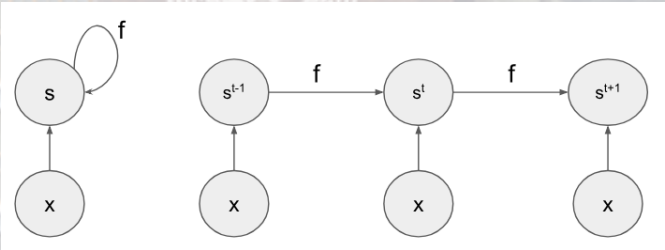
$$s^{(5)} = f(f(f(s^{(1)}; \theta); \theta); \theta)$$



Despliegue de redes

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

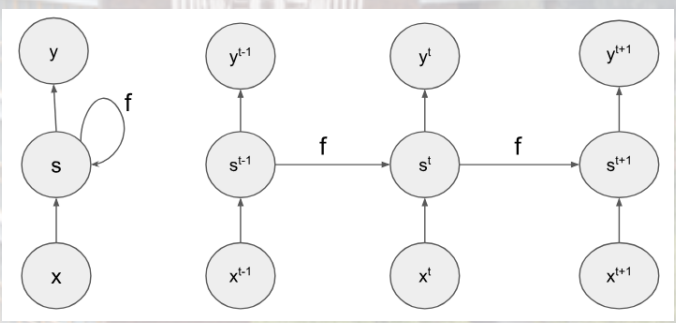
$$s^{(t)} = f\left(s^{(t-1)}, x^{(t)}; \theta\right) \tag{2}$$



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: $x^{(1)}, x^{(2)}, ..., x^{(T)} \mapsto y^{(1)}, y^{(2)}, ..., y^{(T)}$.

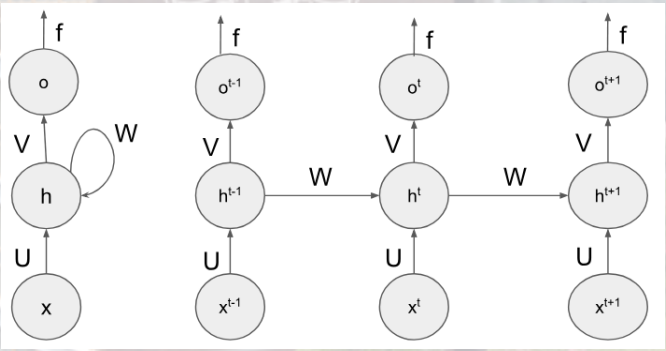
Definimos este tipo de RNNs a partir de los siguientes elementos:

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

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:

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

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 eñ 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:

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

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:

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

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

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

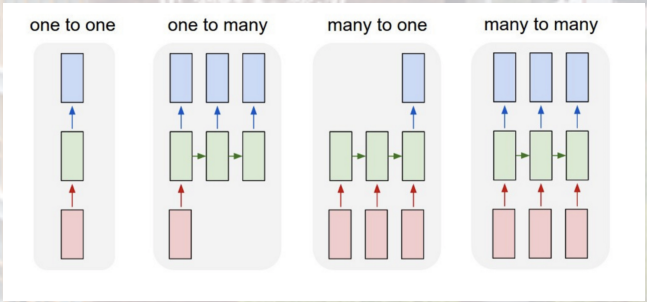
Si la red recurrente tiene capas intermedias entre la entrada y la recurrencia o entre la recurrencia y la entrada estas se actualizarán con el método de backpropagation a través del tiempo.

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:



Otros tipos de RNNs

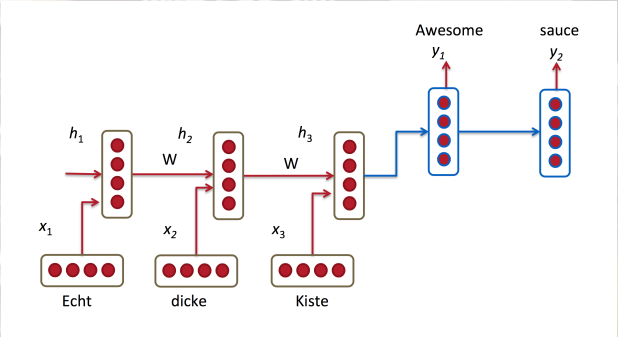
- El modelo one-to-one corresponde a una red neuronal pre-alimentada común.
- En el modelo de one-to-many, se puede tomar la salida del estado anterior como entrada en el estado actual:

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

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”:

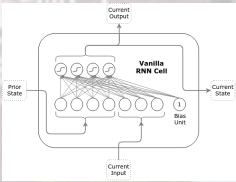


Vanilla RNN y LSTM

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

$$h^{(t)} = g\left(Wh^{(t)} + Ux^{(t)} + b\right)$$



- 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 reelevante y no sobreescribir.
- ¿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:

$$
\begin{aligned}
i_t &= \sigma(W_i h^{(t-1)} + U_i x^{(t)} + b_i) \\
o_t &= \sigma(W_o h^{(t-1)} + U_o x^{(t)} + b_o) \\
f_t &= \sigma(W_f h^{(t-1)} + U_f x^{(t)} + b_f)
\end{aligned}
$$

Y proponemos una variable de lectura:

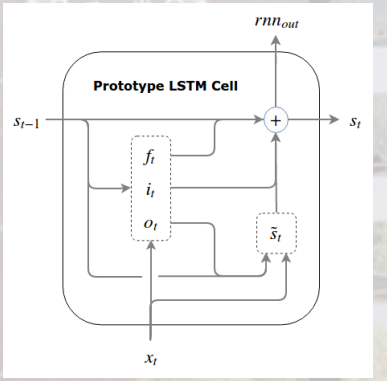
$$\hat{h}_t = \phi(W[o_t \odot h^{(t-1)}] + U x^{(t)} + b) \tag{6}$$

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

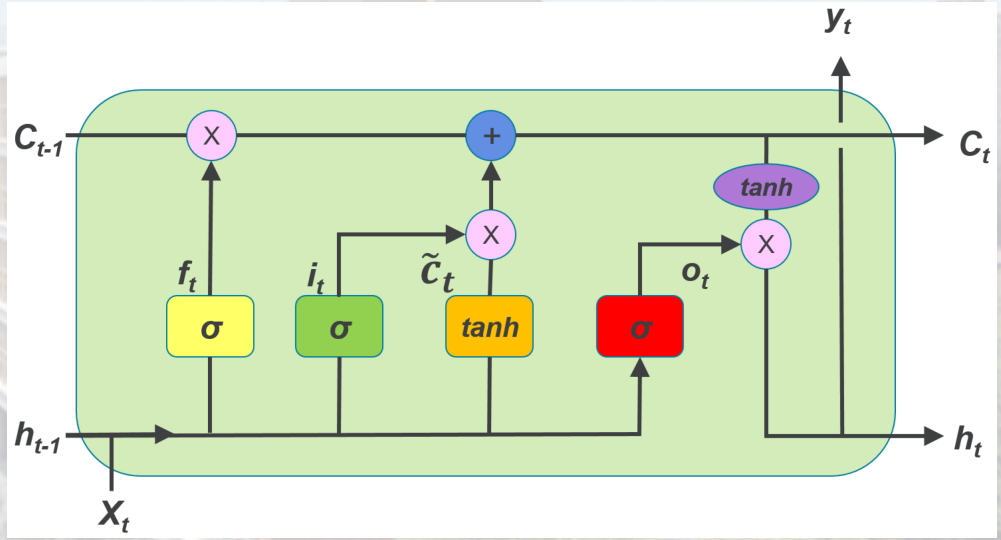
Los valores i_t, o_t y f_t permanecen igual (exceptuando el valor de $h^{(t-1)}$. Definimos:

$$\hat{c}_t = \tanh(Wh^{(t-1)} + Ux^{(t)} + b)$$
$$c_t = f_t \odot c_{t-1} + i_t \odot \hat{c}_t$$

Finalmente:

$$h^{(t)} = o_t \odot \tanh(c_t) \tag{8}$$

LSTM



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.

The End