Deep Learning

Episodio 6: Redes Neuronales Recurrentes

Fernando Gama

Escuela de Graduados en Ingeniería Informática y Sistemas, Facultad de Ingeniería, UBA

11 de Agosto de 2022

Escenas del capítulo anterior...

- Redes Neuronales Convolucionales $\Rightarrow$ Reemplazar la transformación lineal por una convolución
  $\Rightarrow$ Se pierde el control sobre la capacidad de representación
  $\Rightarrow$ Banco de Filtros + Suma

- Pooling $\Rightarrow$ Construir resúmenes regionales y luego quedarse sólo con un representante
  $\Rightarrow$ Controlar el tamaño de las señales $\Rightarrow$ Invarianza local a traslaciones

- Arquitecturas típicas $\Rightarrow$ Incluir MLPs al final, conexiones residuales, etc.

- Consideremos una secuencia de datos $\{\mathbf{x}_0, \mathbf{x}_1, \ldots\}$ con $\mathbf{x}_t \in \mathbb{R}^F$ $\Rightarrow$ Queremos procesarlos
- La suposición de estructura es que estos datos están relacionados secuencialmente
  $\Rightarrow$ $\mathbf{x}_t, \mathbf{x}_{t'}$ están relacionados, de alguna forma, para todo t, t' (puede ser débil si $|t - t'| \gg 1$)
  $\Rightarrow$ No sabemos cuál es esa relación, pero suponemos que existe $\Rightarrow$ Queremos aprenderla (o algo así)

- Para aprender las relaciones de la secuencia, asumimos que existe una variable $\mathbf{h}_t \in \mathbb{R}^H$
  $\Rightarrow$ La secuencia de variables $\{\mathbf{h}_0, \mathbf{h}_1, \ldots\}$ captura la información relevante de $\{\mathbf{x}_t\}$

- La secuencia de variables $\{\mathbf{h}_t\}$ depende de la secuencia de datos $\{\mathbf{x}_t\}$ mediante un sistema dinámico

$$\mathbf{h}_t = \mathsf{f}\big(\mathbf{h}_{t-1}, \mathbf{x}_t\big)$$

- La función f es capaz de producir el nuevo valor de $\mathbf{h}_t$
  - $\Rightarrow$ Conociendo únicamente el valor anterior del estado $\mathbf{h}_{t-1}$
  - $\Rightarrow$ Y el valor actual del dato $\mathbf{x}_t$

- Ciertamente, queremos aprender la función f $\Rightarrow$ Vamos a parametrizarla

$$\mathbf{h}_t = \mathsf{f}\big(\mathbf{h}_{t-1}, \mathbf{x}_t; \boldsymbol{\Theta}\big)$$

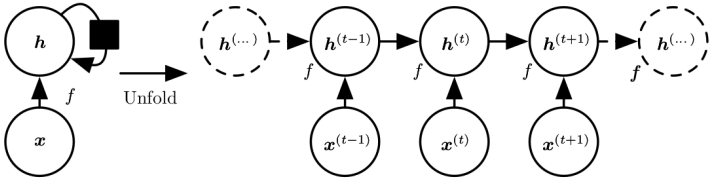
- El valor aprendido de $\mathbf{h}_t$ se conoce como estado oculto (*hidden state*)
- El objetivo es aprender f de tal manera que $\mathbf{h}_t$ capture información relevante para alguna tarea

Desdoblando Grafos

- La relación de recurrencia no parece encajar con la idea de una computación hacia adelante

$$\mathbf{h}_t = \mathsf{f}\big(\mathbf{h}_{t-1}, \mathbf{x}_t\big) = \mathsf{g}\big(\{\mathbf{h}_\tau\}_{\tau=0}^{t-1}, \{\mathbf{x}_\tau\}_{\tau=0}^{t}\big)$$

- Se desdobla el grafo de la computación recurrente $\Rightarrow$ Grafo acíclico $\Rightarrow$ Computación en cada capa

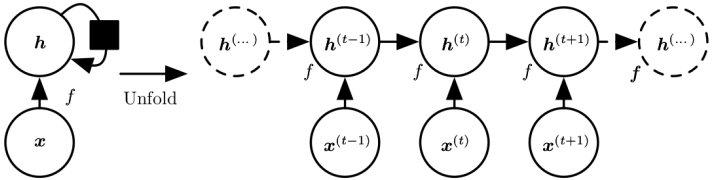


- Si bien la función g tiene una cantidad variable de entradas

$$\mathbf{h}_t = \mathrm{f}\big(\mathbf{h}_{t-1}, \mathbf{x}_t\big) = \mathrm{g}\big(\{\mathbf{h}_\tau\}_{\tau=0}^{t-1}, \{\mathbf{x}_\tau\}_{\tau=0}^{t}\big)$$

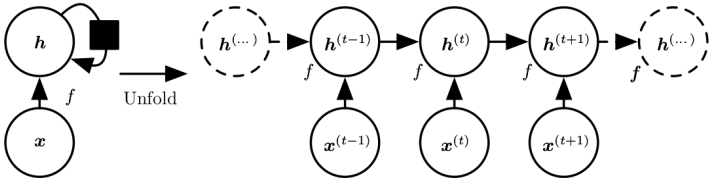
  $\Rightarrow$ Al ser un grafo desdoblado, la función g se construye mediante sucesivas aplicaciones de f

  $\Rightarrow$ La función f que se aprende es una sola, y tiene siempre las mismas entradas: $\mathbf{h}_{t-1}, \mathbf{x}_t$
- Es posible aprender una función, independientemente de la longitud de la secuencia
- Comparten parámetros para todos los valores de t $\Rightarrow$ Generalización



Redes Neuronales Recurrentes

- Una red neuronal recurrente (RNN) es una arquitectura especializada en procesar secuencias de datos
  $\Rightarrow$ Generalizan mejor para secuencias de datos, se pueden aplicar a secuencias de longitud variable
- La clave está en el desdoble de grafos para compartir parámetros
- En algún punto se puede pensar cualquier función recurrente como una RNN

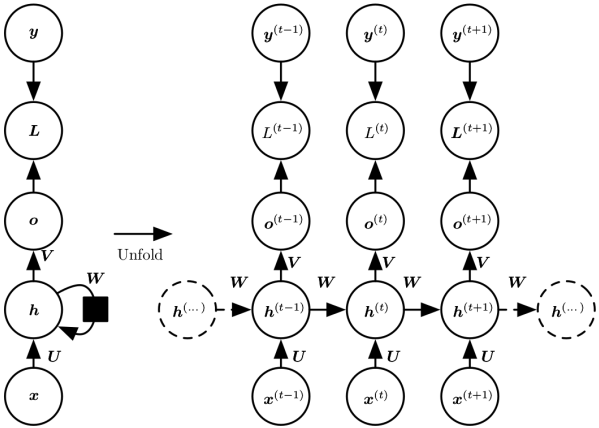


- Vamos a parametrizar con una red neuronal a la función f (una sola capa)

$$\mathbf{h}_t = \sigma\big(\mathbf{W}\mathbf{h}_{t-1} + \mathbf{U}\mathbf{x}_t + \mathbf{b}\big)$$
$$\mathbf{o}_t = \rho\big(\mathbf{V}\mathbf{h}_t + \mathbf{c}\big)$$

- La salida de la RNN no es el estado oculto $\mathbf{h}_t$ pero otro vector $\mathbf{o}_t$
  $\Rightarrow$ Se separa el aprendizaje del estado $\mathbf{h}_t$ del aprendizaje de la salida $\mathbf{o}_t$
- En general, se toma $\sigma(x) = \tanh(x)$, y $\rho(x) = x$ (ninguna no-linealidad a la salida)
- Los parámetros $\boldsymbol{\Theta} = \{\mathbf{W}, \mathbf{U}, \mathbf{b}, \mathbf{V}, \mathbf{c}\}$ son los mismos para todo instante de tiempo t
  $\Rightarrow$ Compartir los parámetros permite generalizar mejor cuando procesamos datos secuenciales

Redes Neuronales Recurrentes

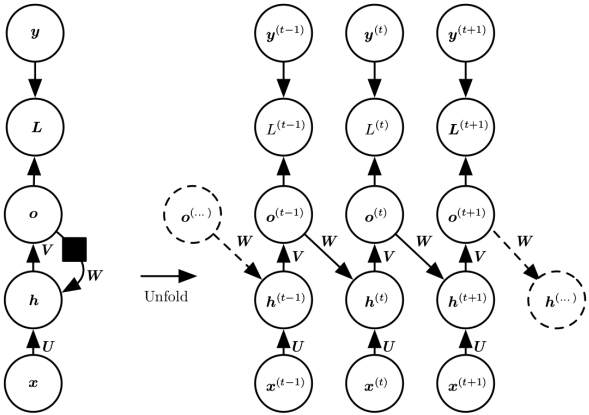


$$\mathbf{h}_t = \tanh\left(\mathbf{W}\mathbf{h}_{t-1} + \mathbf{U}\mathbf{x}_t + \mathbf{b}\right)$$
$$\mathbf{o}_t = \mathbf{V}\mathbf{h}_t + \mathbf{c}$$

- Mapea una secuencia de entrada $\{\mathbf{x}_t\}$
  $\Rightarrow$ En una de salida $\{\mathbf{o}_t\}$
- La función de pérdida viene dada por

$$J_t = \mathsf{J}\left(\mathbf{y}_t, \mathbf{o}_t\right) = \mathsf{J}\left(\mathbf{y}_t, \{\mathbf{x}_\tau\}_{\tau=0}^{t}\right)$$

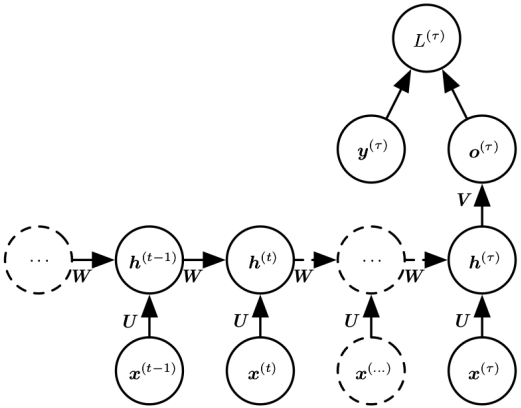
Redes Neuronales Recurrentes



$$\mathbf{h}_t = \tanh\left(\mathbf{W}\mathbf{o}_{t-1} + \mathbf{U}\mathbf{x}_t + \mathbf{b}\right)$$
$$\mathbf{o}_t = \mathbf{V}\mathbf{h}_t + \mathbf{c}$$

- Menor capacidad de representación
  - $\Rightarrow \mathbf{o}_t$ doble función de ser salida
  - $\Rightarrow$ Y conservar información
- Más fácil de entrenar (paralelización)

Redes Neuronales Recurrentes



$$\mathbf{h}_t = \tanh\left(\mathbf{W}\mathbf{h}_{t-1} + \mathbf{U}\mathbf{x}_t + \mathbf{b}\right)$$
$$\mathbf{o}_T = \mathbf{V}\mathbf{h}_T + \mathbf{c}$$

- Igual que la primera
  - $\Rightarrow$ Pero con una única salida
  - $\Rightarrow$ A tiempo T
- Funciona para resumir secuencias

Entrenamiento de RNNs

- La RNN estándar es la primera que vimos $\Rightarrow$ Es la que vamos a usar de ahora en más

$$\mathbf{h}_t = \sigma\big(\mathbf{W}\mathbf{h}_{t-1} + \mathbf{U}\mathbf{x}_t + \mathbf{b}\big)$$
$$\mathbf{o}_t = \rho\big(\mathbf{V}\mathbf{h}_t + \mathbf{c}\big)$$

- El vector de estado oculto $\mathbf{h}_t$ debe capturar información relevante para el problema a resolver

- Calcular el gradiente es costoso $\Rightarrow$ Hay que moverse a través del tiempo
  - $\Rightarrow$ Y luego hay que volver para atrás
  - $\Rightarrow$ El costo es $\mathsf{O}(T)$ (longitud de la secuencia)
- No se puede reducir mediante paralelización $\Rightarrow$ La pasada hacia adelante es secuencial
  - $\Rightarrow$ Cada instante de tiempo necesita el anterior (como las capas)
- El costo en memoria también es $\mathsf{O}(T)$ $\Rightarrow$ Hay que guardar los estados anteriores

Entrenamiento de RNNs

- Para calcular el gradiente, simplemente hacemos *backpropagation* de manera usual
  $\Rightarrow$ Pensar cada instante de tiempo como una capa diferente
- En el contexto de RNNs esto se conoce como BPTT (*backpropagation through time*)

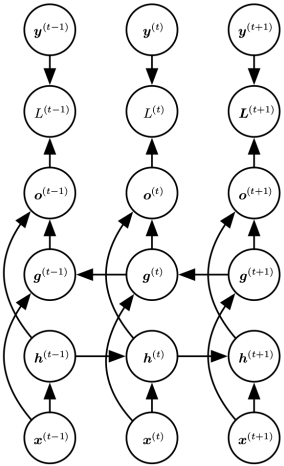
- Sin embargo, los valores de los parámetros $\boldsymbol{\Theta} = (\mathbf{W}, \mathbf{U}, \mathbf{b}, \mathbf{V}, \mathbf{c})$ se comparten a través de t
  $\Rightarrow$ Suponemos, entonces, que $\boldsymbol{\Theta}$ depende del tiempo $\boldsymbol{\Theta}_t$ y forzamos $\boldsymbol{\Theta}_t = \boldsymbol{\Theta}$ para todo t
  $\Rightarrow$ Luego, cada vez que aparece $\boldsymbol{\Theta}_t$ en el gradiente, reemplazamos por $\boldsymbol{\Theta}$
- Queda una suma para todos los instantes de tiempo (como solía ser para todas las capas)
- Tenemos sólo 5 gradientes que calcular $\Rightarrow$ Ver ecuaciones (10.22) a (10.28) para un ejemplo

Extensiones

Sistemas Dinámicos

Redes Neuronales Recurrentes

Extensiones



- Si el contexto importa, tenemos que saber qué hay alrededor del valor
- Crear una RNN que vaya para adelante, en valores de t crecientes
- Y otra que vaya para atrás, en valores de t decrecientes

$$\mathbf{h}_t = \sigma_f\left(\mathbf{W}_f \mathbf{h}_{t-1} + \mathbf{U}_f \mathbf{x}_t + \mathbf{b}_f\right)$$
$$\mathbf{g}_t = \sigma_b\left(\mathbf{W}_b \mathbf{g}_{t+1} + \mathbf{U}_b \mathbf{x}_t + \mathbf{b}_b\right)$$
$$\mathbf{o}_t = \rho\left(\mathbf{V}_f \mathbf{h}_t + \mathbf{V}_b \mathbf{g}_t + \mathbf{c}\right)$$

- Podemos pensar a las RNNs como una función de la entrada al estado oculto
  - $\Rightarrow$ Otra función del viejo estado oculto al nuevo estado oculto
  - $\Rightarrow$ Y otra función del estado oculto a la salida

$$\mathbf{h}_t = \mathsf{f}_{h \to h}\left(\mathbf{h}_{t-1}\right) + \mathsf{f}_{x \to h}\left(\mathbf{x}_t\right)$$
$$\mathbf{o}_t = \mathsf{f}_{h \to o}\left(\mathbf{h}_t\right)$$

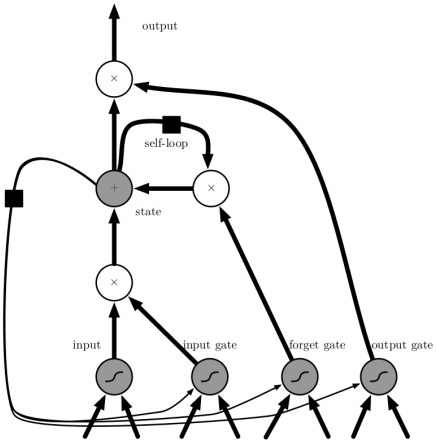
- Cada una de estas se puede reemplazar por una MLP arbitraria
  - $\Rightarrow$ En la formulación original usamos una MLP de una única capa
  - $\Rightarrow$ Usar muchas capas $\Rightarrow$ Ganar en profundidad $\Rightarrow$ Mayor capacidad de representación
- Se ha observado en la práctica que RNNs más profundas mejoran el desempeño
- Tener en cuenta que el problema de optimización se vuelve más difícil

Dependencias de Largo Alcance

- Cuando tenemos secuencias muy largas, la información del principio se disipa
- Los gradientes se desvanecen (la mayoría de las veces) o explotan (raramente)
- Incluso si se pudieran controlar los pesos, la información del pasado decrece exponencialmente
  - $\Rightarrow$ Esto depende de los autovalores de las matrices $\Rightarrow$ Ej., $\mathbf{W}^t$ depende de λ_i^t
  - $\Rightarrow$ Los autovalores menores que uno, desaparecen, los mayores que uno, explotan
- La información contenida en el pasado es tan pequeña que queda enmascarada por el ruido

- El desafío está en lograr aprender utilizando información del pasado
  - $\Rightarrow$ ¿Cómo se incorporan las dependencias de largo alcance?

LSTM: *Long Short-Term Memory*



- Se incorporan tres compuertas que controlan el flujo
  - ⇒ Input gate: si el dato pasa o no
  - ⇒ Forget gate: la importancia del estado oculto
  - ⇒ Output gate: cuánto del estado usar en la salida

$$\mathbf{f}_t = \sigma_f\left(\mathbf{W}_f\mathbf{h}_{t-1} + \mathbf{U}_f\mathbf{x}_t + \mathbf{b}_f\right) \quad \text{forget gate}$$

$$\mathbf{g}_t = \sigma_g\left(\mathbf{W}_g\mathbf{h}_{t-1} + \mathbf{U}_g\mathbf{x}_t + \mathbf{b}_g\right) \quad \text{input gate}$$

$$\mathbf{q}_t = \sigma_o\left(\mathbf{W}_o\mathbf{h}_{t-1} + \mathbf{U}_o\mathbf{x}_t + \mathbf{b}_o\right) \quad \text{output gate}$$

$$\mathbf{s}_t = \mathbf{f}_t \odot \mathbf{s}_{t-1} + \mathbf{g}_t \odot \sigma\left(\mathbf{W}\mathbf{h}_{t-1} + \mathbf{U}\mathbf{x}_t + \mathbf{b}\right) \quad \text{estado interno}$$

$$\mathbf{h}_t = \mathbf{q}_t \odot \tanh(\mathbf{s}_t) \quad \text{estado oculto}$$

GRU: *Gated Recurrent Unit*

- Simplificar las LSTM usando menos gates
  - $\Rightarrow$ Update gate: cuánto actualizar el estado
  - $\Rightarrow$ Reset gate: qué partes del estado utilizar

$$\mathbf{u}_t = \sigma_u\big(\mathbf{W}_u\mathbf{h}_{t-1} + \mathbf{U}_u\mathbf{x}_t + \mathbf{b}_u\big) \qquad \text{update gate}$$
$$\mathbf{r}_t = \sigma_r\big(\mathbf{W}_r\mathbf{h}_{t-1} + \mathbf{U}_r\mathbf{x}_t + \mathbf{b}_r\big) \qquad \text{reset gate}$$
$$\mathbf{s}_t = \mathbf{u}_t \odot \mathbf{h}_{t-1} + (1 - \mathbf{u}_t) \odot \sigma\big(\mathbf{W}(\mathbf{r}_t \odot \mathbf{h}_{t-1}) + \mathbf{U}\mathbf{x}_t + \mathbf{b}\big) \qquad \text{estado}$$

- Menos parámetros para aprender $\Rightarrow$ Más fáciles de optimizar

Optimización de RNNs

- Si los gradientes son muy grandes
  - $\Rightarrow$ Si dan `Inf` o `Nan` se puede tomar un paso en cualquier dirección para salir de ahí
  - $\Rightarrow$ Si son muy, muy grandes, se puede hacer *gradient clipping*
  - $\Rightarrow$ Donde se mantiene la dirección del gradiente original, pero se achica su norma

- En el caso de gradientes pequeños se puede entrenar con una regularización

$$\sum_t \left(\frac{\left\| \frac{\partial L}{\partial \mathbf{h}_t} \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t-1}} \right\|}{\left\| \frac{\partial L}{\partial \mathbf{h}_t} \right\|} - 1 \right)^2$$

  - $\Rightarrow$ Se fuerza a que la información de $\mathbf{h}_{t-1}$ a $\mathbf{h}_t$ sea grande
- Si se tienen muchos datos, usar LSTM puede resultar en mejor desempeño

- Procesar secuencias de datos sin usar RNNs
- Calcular la *attention* de todos los elementos

  $\Rightarrow$ Aprender qué parte de la secuencia es más importante

$$\mathbf{q}_t = \mathbf{W}_q \mathbf{x}_t \qquad \textit{query}$$
$$\mathbf{k}_t = \mathbf{W}_k \mathbf{x}_t \qquad \textit{key}$$
$$\mathbf{v}_t = \mathbf{W}_v \mathbf{x}_t \qquad \textit{value}$$
$$\mathbf{Q}, \mathbf{K}, \mathbf{V} \qquad \text{matrices}$$
$$\mathbf{Y} = \mathsf{softmax}\left(d_k^{-1/2} \mathbf{Q}^\mathsf{T} \mathbf{K}\right)\mathbf{V} \qquad \textit{attention}$$

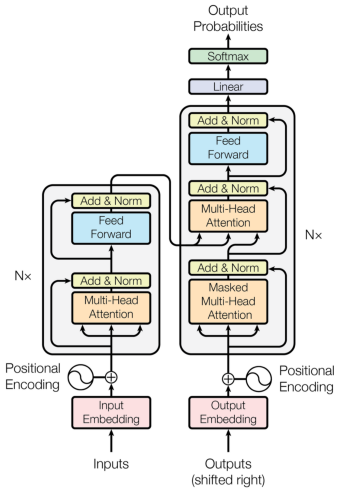


Figure 1: The Transformer - model architecture.

Resumen del Capítulo

- Queremos procesar secuencias de datos
  - $\Rightarrow$ Adaptamos la red neuronal para generalizar en datos secuenciales
  - $\Rightarrow$ Compartir parámetros $\Rightarrow$ Mismos valores durante toda la secuencia

- Redes neuronales recurrentes $\Rightarrow$ Aprenden un estado oculto
  - $\Rightarrow$ Este estado es capaz de capturar la información temporal relevante

- Área activa de investigación $\Rightarrow$ Muchas extensiones
  - $\Rightarrow$ Redes neuronales bidireccionales $\Rightarrow$ El contexto importa
  - $\Rightarrow$ LSTMs, GRUs $\Rightarrow$ Poder aprender dependencias de largo alcance

Escenas del Próximo Capítulo...

- La importancia de los datos con estructuras descriptas por grafos
  $\Rightarrow$ Requieren tener en cuenta el grafo para procesar de manera adecuada

- Señales en grafos, operador de desplazamiento en el grafo
  $\Rightarrow$ Convolución en grafos $\Rightarrow$ Combinación lineal de señales en el vecindario

- Redes neuronales en grafos $\Rightarrow$ Reemplazar la operación lineal por una convolución en el grafo
  $\Rightarrow$ Explota la estructura de los datos descripta por el grafo