uc3m | Universidad Carlos III de Madrid

OpenCourseWare

Procesamiento de Lenguaje Natural con Aprendizaje Profundo,

Máster en Ciencia y Tecnología Informática

Tema 3: Redes recurrentes para Procesamiento de Lenguaje Natural

Isabel Segura Bedmar

Objetivos

- Conocer los fundamentos de las redes recurrentes (en inglés, recurrent neural networks (RNN)).
- Conocer los tipos de redes recurrentes y las características de sus neuronas.
- Aprender a usar redes recurrentes para resolver tareas de PLN.

Índice

- Redes recurrentes
- Otros ejemplos de aplicaciones de RNN en PLN
- Tipos de RNN
  - simple
  - LSTM
  - GRU

Redes recurrentes (RNN)

- Dada una secuencia de palabras, el modelo debe **predecir la siguiente palabra**:

El *gobierno* *aprobó* *un* **?**

$x_{(1)}$ $x_{(2)}$ $x_{(3)}$ $x_{(4)}$

decreto

plan

acuerdo

aumento

- Una **red recurrente** es una red neuronal pensada para **procesar secuencias** de datos $\mathbf{x} = (x_{(1)}, x_{(2)}, .., x_{(n)})$.
- Cuando se procesa un dato de la secuencia, la red usa como entrada las salidas de los datos anteriores en la secuencia.

Redes recurrentes (RNN)

- Cada **token** es asociado con un número que es el **índice** de la palabra en el **vocabulario**, **V**, (conjunto de tokens únicos del conjunto de entrenamiento).
- Dicho número se puede representar como un **vector one-hot**, $\mathbf{x}_{(j)} \in \mathbf{R}^{|V|}$. En dicho vector, todos sus elementos son 0, excepto el elemento cuyo índice coincide con el índice del token en V).
- Por ejemplo, si |V| = 9 y el índice de token “decreto” en V es 5. Entonces su vector one-hot será:

0	0	0	0	1	0	0	0	0
1	2	3	4	5	6	7	8	9

Redes recurrentes (RNN)

- Creamos una **matriz de embeddings**, **E**, de dimensión **|V| x d**, donde d es la dimensión de los embeddings.
- Cada **fila** de la matriz es un **embedding** de un token de V. El **índice de la fila** es el **índice del token** en el V.
- Dimensiones frecuentes de embeddings: 50, 100, 200 o 300.
- **E** puede ser **inicializada** de forma **aleatoria** o creada a partir de un **modelo de Word Embedding** (tema 4).
- Algunos métodos de inicialización aleatoria son:
  - zero-initialization
  - random initialization (siguiendo por ejemplo una distribución normal o uniforme.
  - xavier initialization
  - he-et-al initialization

Redes recurrentes (RNN)

- Por ejemplo, si |V| = 9 y d = 4 (dimensión de los embeddings), una posible matriz **E** (valores aleatorios con distribución uniforme):

```
1 import math
2 import numpy as np
3 np.random.seed(0)
4 scale = 1/max(1., (2+2)/2.)
5 limit = math.sqrt(3.0 * scale)
6 weights = np.random.uniform(-limit, limit, size=(9,4))
7 print(weights)
```

d

|V|

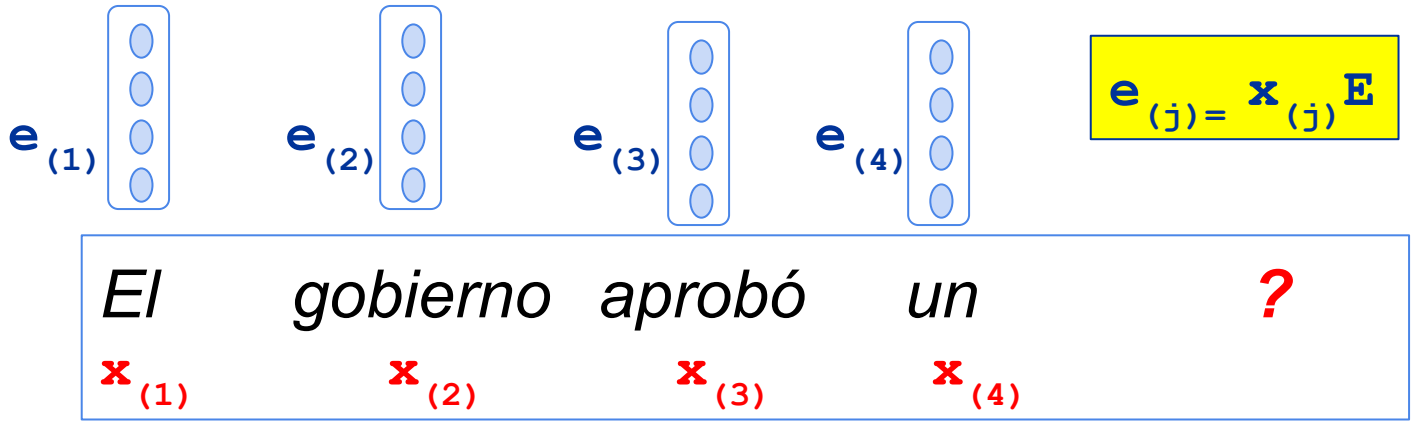
```
[[ 0.11956818  0.52710415  0.25171784  0.1099409 ]
 [-0.18700679  0.35736613 -0.15287949  0.95964395]
 [ 1.13573718 -0.2855088   0.71457749  0.07077781]
 [ 0.16667445  1.0424946  -1.05074278 -1.01132255]
 [-1.17522011  0.8147489   0.68134211  0.90634096]
 [ 1.17237072  0.73278583 -0.09435591  0.68715334]
 [-0.93503288  0.34273511 -0.87360246  1.08921195]
 [ 0.05351724 -0.2090347  -0.57671861  0.67173261]
 [-0.10740931  0.16762826 -1.17871945  0.28814694]]
```

d

embedding para el token cuyo índice es 5 (“decreto”)

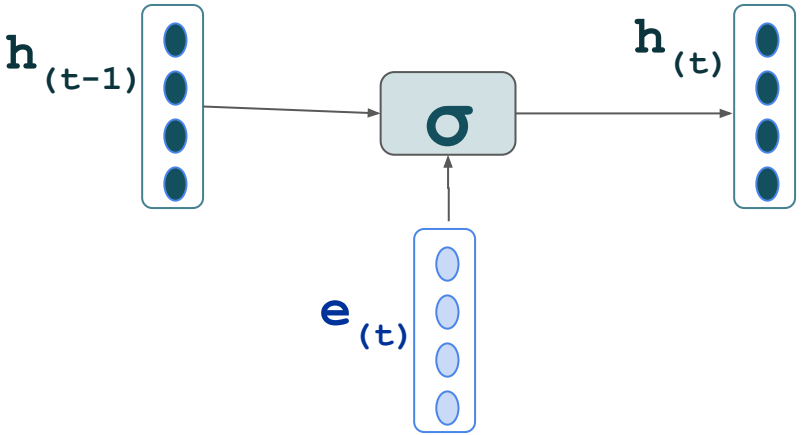
Redes recurrentes (RNN)

- La **capa de entrada** de la red son los **embeddings** de los tokens.
- El embedding para cada token se obtiene multiplicando su vector one-hot y la matriz E:

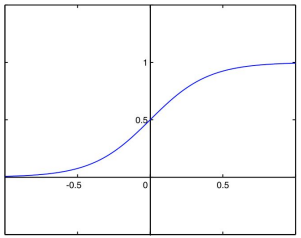


Redes recurrentes (RNN)

- En una RNN, después de la capa de entrada (capa de entrada), pueden existir una o varias **capas ocultas**.
- La capa oculta produce un **estado oculto,** $h_{(t)}$, por cada **embedding,** $e_{(t)}$, de la capa anterior.
- Para calcular el estado $h_{(t)}$, una **función de activación**, por ejemplo, tangente hiperbólica (tanh) o sigmoide (σ) es aplicada sobre el **estado anterior** $h_{(t-1)}$ y el **embedding** $e_{(t)}$.

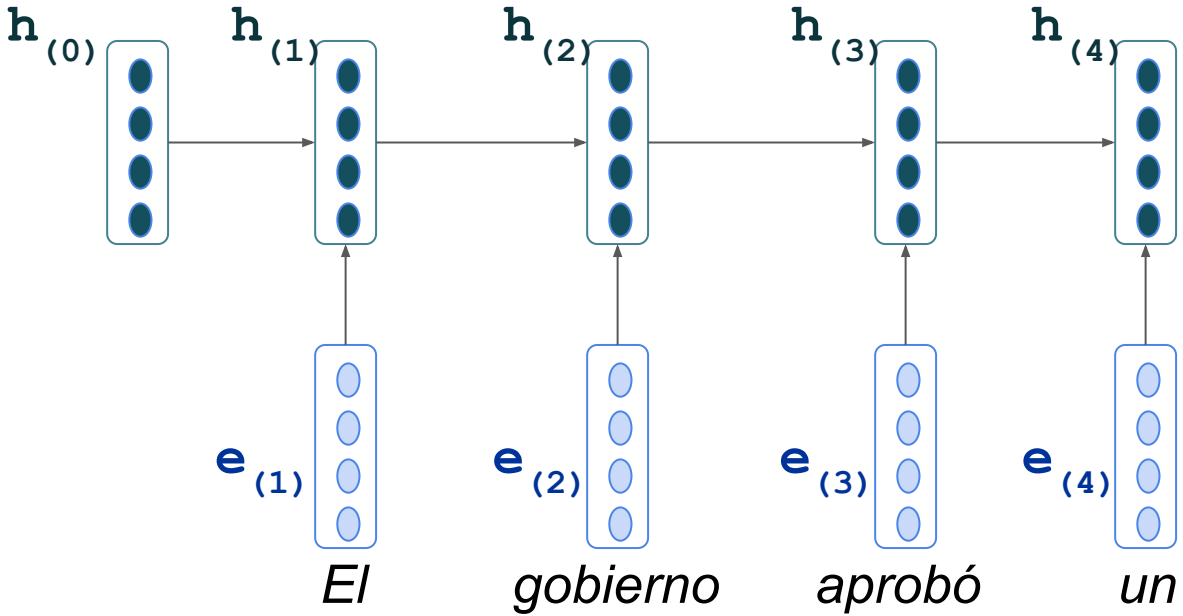


$$\sigma(x) = \frac{1}{1 + e^{-x}}$$



Redes recurrentes (RNN)

- Debemos añadir un **estado inicial,** $\mathbf{h}_{(0)}$, para poder calcular el primer estado (para el primer embedding).

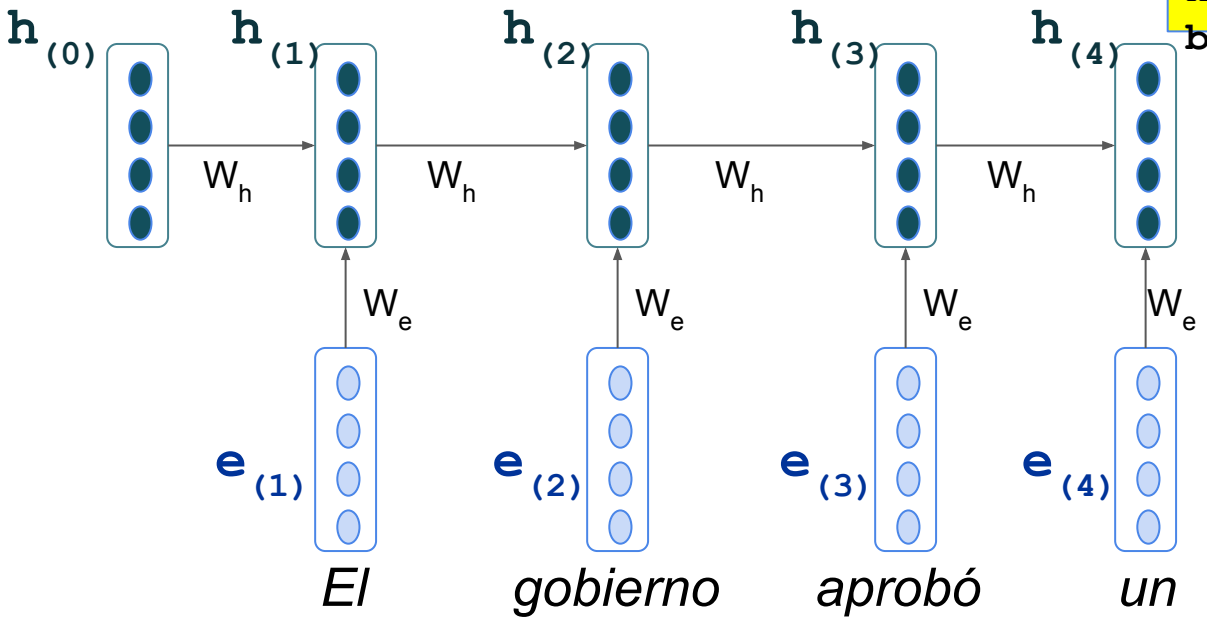


Nota: Para simplificar la figura, estamos obviando las neuronas con las funciones de activación

Redes recurrentes (RNN)

- W_h, W_e: son los pesos (parámetros) de la red, que van a ser ajustados durante el entrenamiento

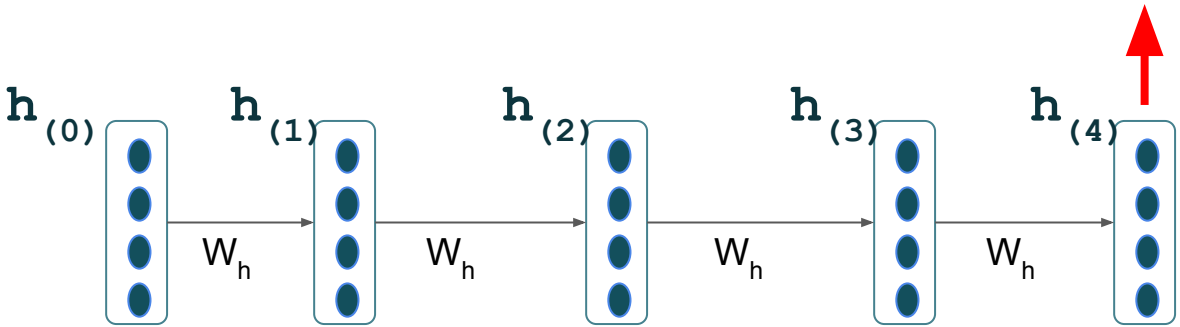
$$h_{(t)} = \sigma(W_h h_{(t-1)} + W_e e_{(t)} + b1)$$



Nota: Para simplificar la figura, estamos obviando las neuronas con las funciones de activación

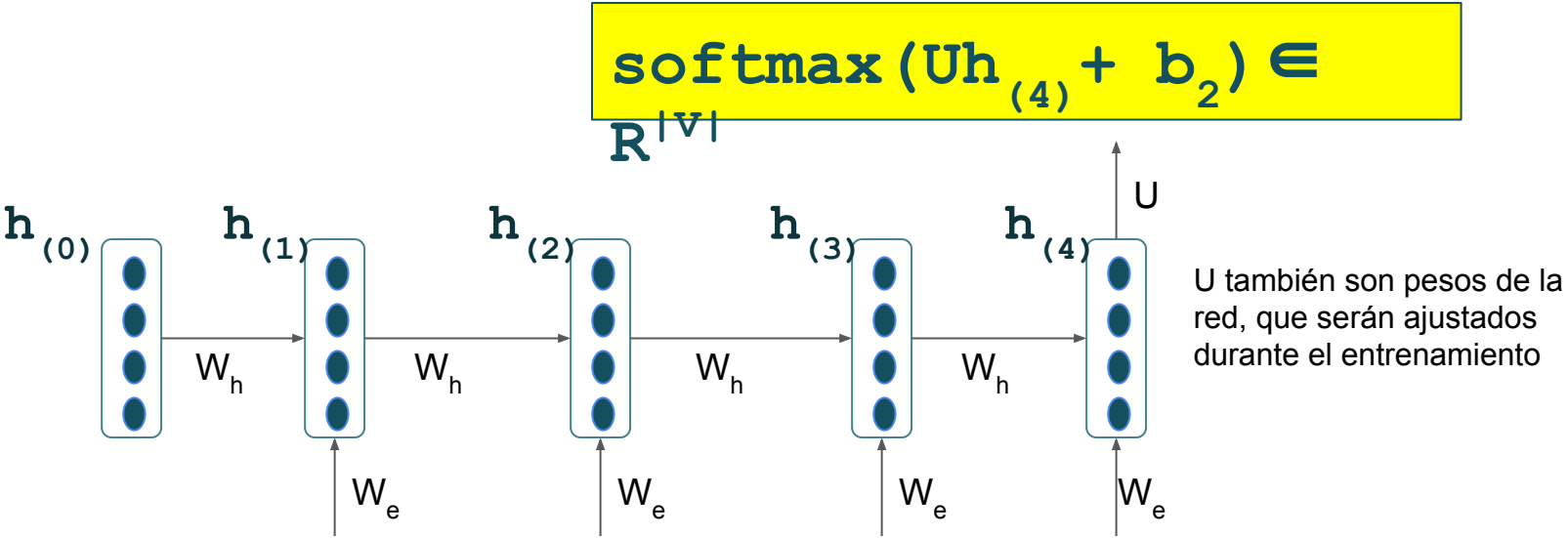
Redes recurrentes (RNN)

- En nuestro ejemplo, el objetivo de la red es predecir el siguiente token que podría completar la secuencia de entrada "*El gobierno aprobó un*".
- La **capa de salida** debe proponer el token que maximice la siguiente probabilidad: **P($x_{(5)}$ | "El gobierno aprobó un ")**



Redes recurrentes (RNN)

- La **entrada** de la **capa de salida** es el **último estado,** en nuestro ejemplo $h_{(4)}$.
- La **capa de salida** utiliza la **función softmax**, que se encarga de transformar los valores del último estado en probabilidades.



Redes recurrentes (RNN)

- Por ejemplo, dado el siguiente último estado, softmax produce las siguientes probabilidades:

0,1
0,000001
0,05
0,0000001
0,9
0,0000001
0,2
0,0000001
0,0000001

→

$$softmax(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}}$$

donde n=|V|

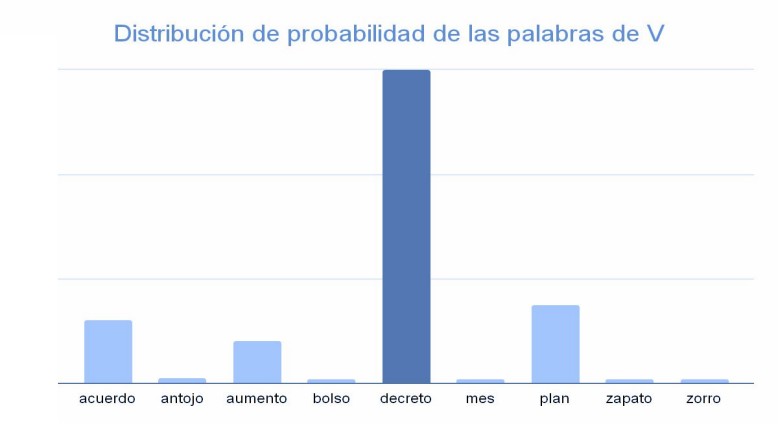
→

0,101
0,092
0,0976
0,092
0,226
0,0923
0,112
0,092
0,092
1

La suma de las probabilidades es 1

Redes recurrentes (RNN)

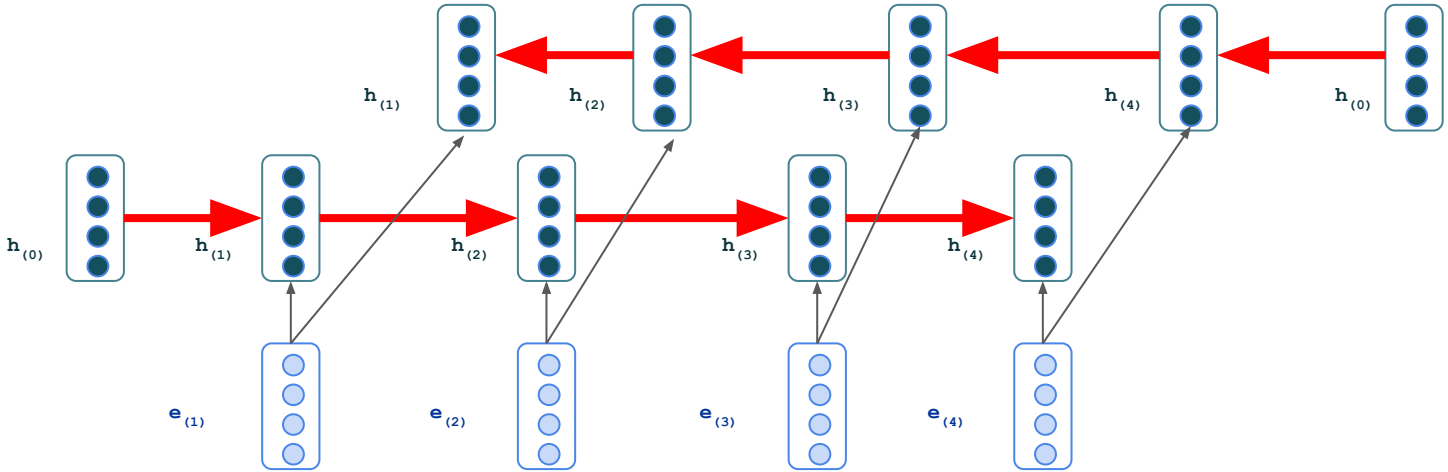
- Es decir, la **función softmax** devuelve un vector de tamaño |V|, donde cada valor se corresponde con la probabilidad de un token en el V.
- Finalmente, la red devuelve el token con mayor probabilidad (en nuestro ejemplo, **decreto**)



El gobierno aprobó un ***decreto***

Redes recurrentes (RNN) - bidireccionales

- En una RNN, la información va pasando de estado en estado, desde el estado inicial hasta el último estado (último token).
- En otras tareas distintas a la predicción del siguiente token, siempre es recomendable **añadir** una **capa** que pueda **leer la secuencia de derecha a izquierda** (desde el último token al primero).



Índice

- Redes recurrentes
- **Otros ejemplos de aplicaciones de RNN en PLN**
- Tipos de RNN
  - básica (vanilla)
  - LSTM
  - GRU

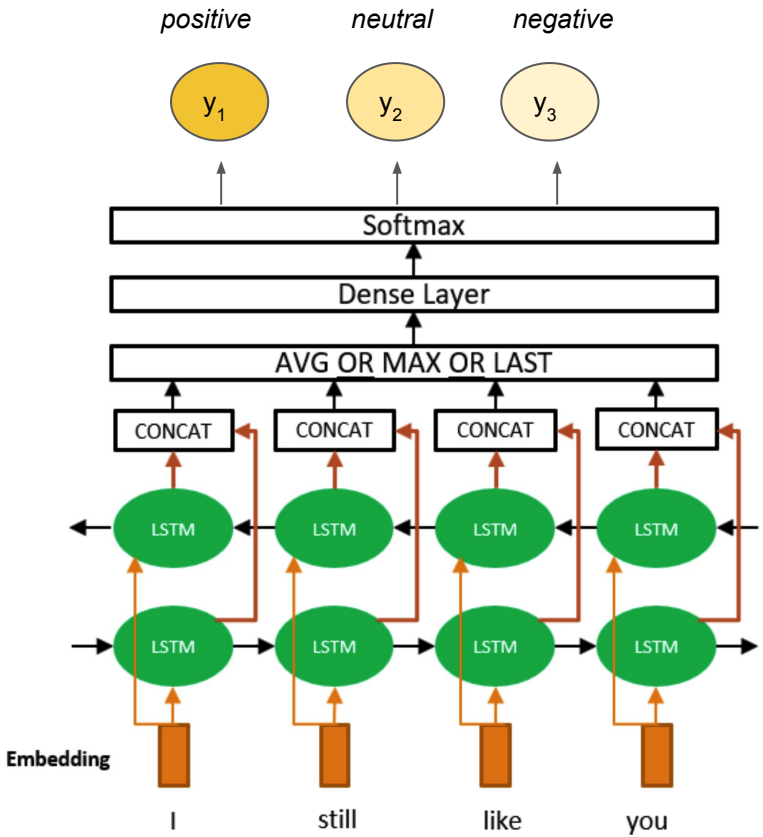
Subtitulado de imágenes

- Recibe una imagen y devuelve un texto que es la descripción de la imagen. Es un modelo que recibe una imagen, y produce una secuencia de tokens como salida.

Imagen de Mohamed Chermiti en Pixabay

Mujer pelirroja toma una fotografía con una cámara PENTAX.

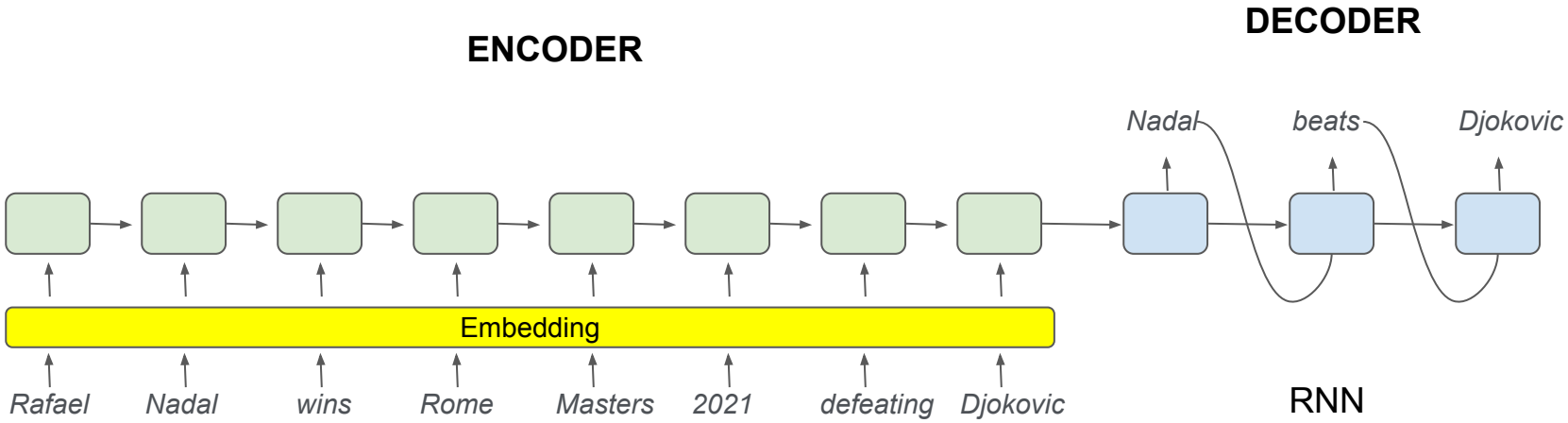
Clasificación de textos



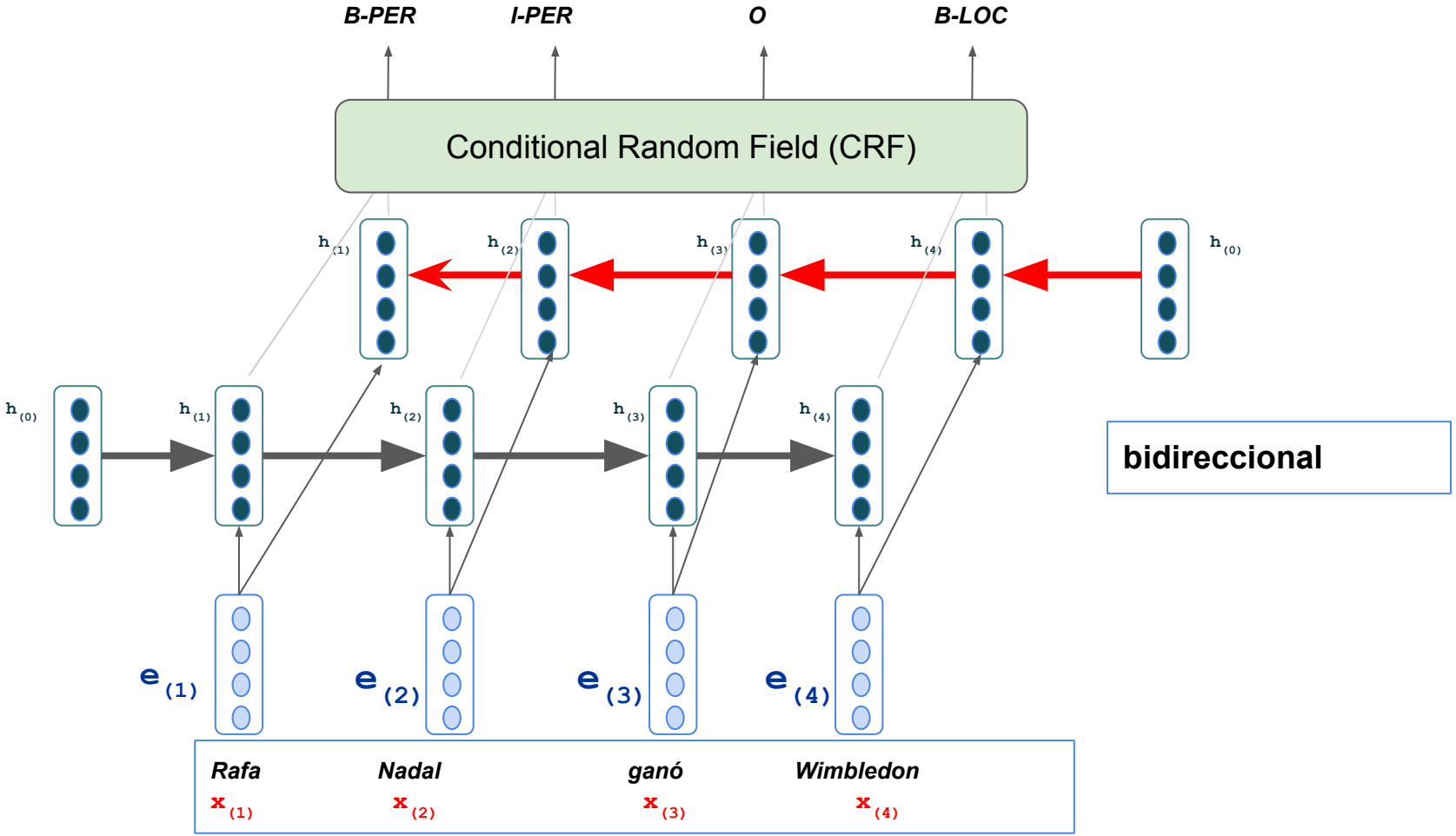
Saroufim, C., Almatarky, A., & Hady, M. A. (2018, October). Language independent sentiment analysis with sentiment-specific word embeddings. In *Proceedings of the 9th Workshop on Computational Approaches to Subjectivity, Sentiment and Social Media Analysis* (pp. 14-23), doi = 10.18653/v1/W18-6204

Generación de resúmenes, traducción automática, etc

- Se consideran modelos Seq2Seq (la entrada es una secuencia, y la salida es otra secuencia). Seq2Seq está formada por un codificador (lee la entrada y genera un representación intermedia) y el decodificador (recibe la salida del codificador y genera la salida).



Reconocimiento de Entidades



Índice

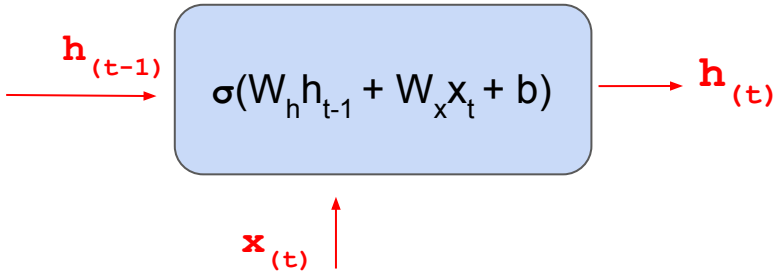
- Redes recurrentes
- Otros ejemplos de aplicaciones de RNN en PLN
- Tipos de RNN
  - básica (vanilla)
  - LSTM
  - GRU

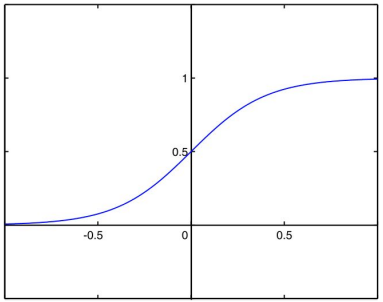
Tipos de RNN

- Clasificación basada en la estructura de la neurona.
  - **básica (vanilla)**: únicamente usa la función **tangente hiperbólica** o **sigmoide**.
  - LSTM: combina distintas funciones de activación para definir tres puertas para controlar el flujo de la información.
  - GRU: también combina distintas funciones de activación aunque únicamente tiene dos puertas.

Tipos de RNN: básica (vanilla)

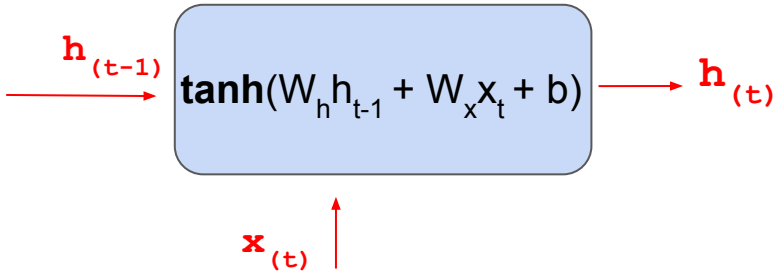
- Únicamente usa la **tangente hipérbolica** o **sigmoide** (como hemos visto en las slides anteriores).
- **sigmoide** garantiza que todo los valores están entre [0,1]. También evita que existan valores con valores mucho mayores que el resto.

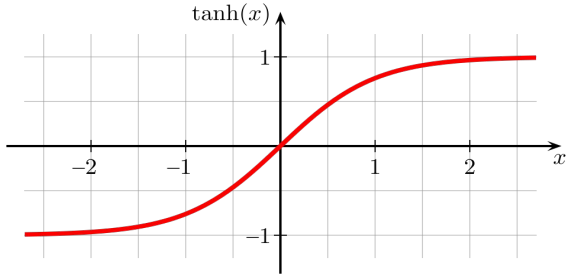




Tipos de RNN: básica (vanilla)

- Del mismo modo, **tanh** también asegura que todos los valores de los vectores siempre estén en [-1,1]. Esto evita que existan valores extremadamente pequeños respecto al resto.

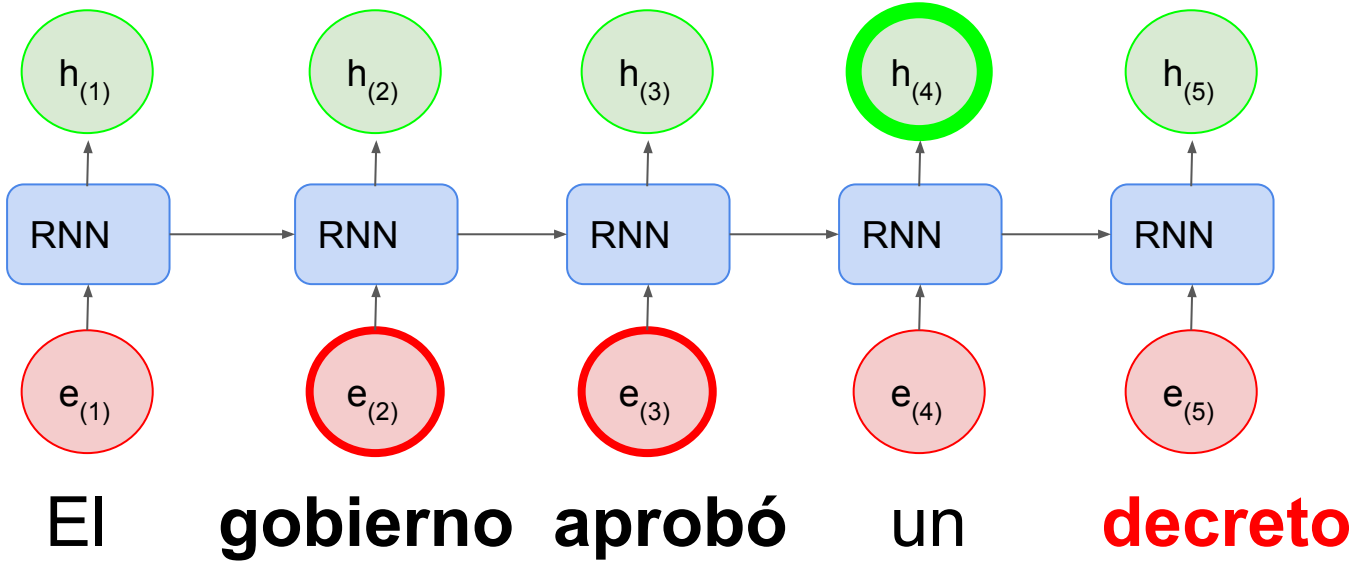




RNN básica (vanilla): problemas

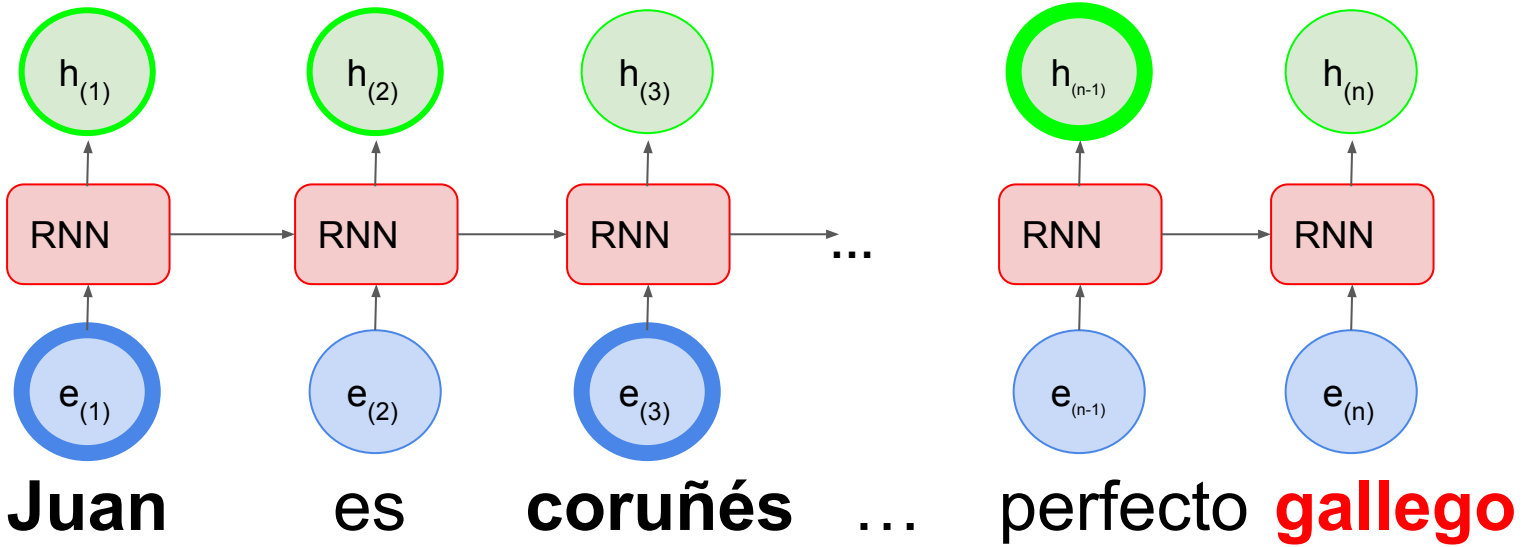
- Aunque las RNN han demostrado ser útiles el procesamiento de secuencias, tienen algunas limitaciones:
  - N**o es capaz de procesar correctamente secuencias largas**. La red tiende a olvidar la información de las primeras neuronas, y no es capaz capturar las relaciones entre elementos distantes. Veamos dos ejemplos:
    - *"**El gobierno** aprobó un **decreto**"*
    - *"**Mi padre** nació en **Pontevedra**. Se crió en un barrio muy humilde, y durante sus primeros años trabajó como chico de los recados, antes de emigrar a Madrid. Él habla un perfecto **gallego**."*

Es es una secuencia corta. Es probable que la información relevante necesaria para predecir “decreto” (se encuentra en los primeros tokens) llega al último estado. La red será capaz de predecir la palabra correcta.



Sin embargo, en este ejemplo, es probable que el último estado apenas almacene información de los primeros estados. La red no tendrá suficiente información para predecir el token correcto.

***Juan** es **coruñés**. Se crió en un barrio muy humilde, y durante sus primeros años trabajó como chico de los recados, antes de emigrar a Madrid. Él habla un perfecto **gallego**.*

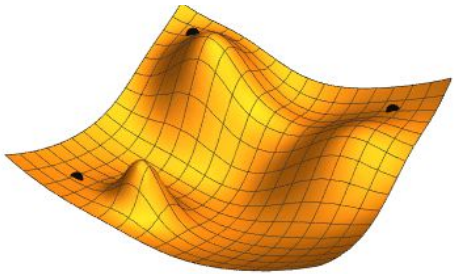


RNN básica (vanilla): problemas

- El segundo de los problemas de las redes RNN básicas es el **desvanecimiento del gradiente** (en inglés *vanishing gradient descent)*.
- El **desvanecimiento del gradiente** ocurre cuando el **gradiente** va tomando **valores muy pequeños**. Esto provoca que los los **parámetros no se modifiquen eficazmente**.

Gradiente descendiente

- Durante la fase de entrenamiento, el **objetivo** es aprender los **parámetros** (pesos) W que **minimizan la función de coste** (loss), es decir, debemos encontrar un mínimo.



Jacopo Bertolotti, CC0, via Wikimedia Commons

- Para ello la red utiliza el método **gradiente descendiente** (*gradient descendent).* Es un un método de optimización numérica para estimar los mejores parámetros.

Gradiente descendiente

- El **gradiente** es el conjunto de todas las **derivadas parciales** de una función.
- En el método de **gradiente descendiente**, se calculan las derivadas parciales de la **función de coste (loss).** La función de coste es el **error cuadrático medio** de las predicciones de la red y los valores en el dataset.

W: parámetros de la red

m: número de ejemplos

$$J(W) = \frac{1}{2m} \sum_{i=1}^{m} [h_W(x_i) - y_i]^2$$

$h_w(x_i)$: predicción para x_i

y_i: valor para x_i en el dataset

Gradiente descendiente

- Una vez calculadas las derivadas parciales, cada parámetro es actualizado con la siguiente ecuación:

α: ratio de aprendizaje (learning ratio)

$$W_i = W_i - \alpha \frac{\partial}{\partial W_i} J(W)$$

- Es decir, cada parámetro es actualizado proporcionalmente a la derivada parcial de la función de coste con respecto al parámetro actual, en cada iteración del entrenamiento.

Gradiente descendiente

α: ratio de aprendizaje (learning ratio)

$$W_i = W_i - \alpha \frac{\partial}{\partial W_i} J(W)$$

- A **mayor gradiente**, **mayor** será el **cambio en W_i**. Por el contrario, a **menor gradiente**, **menor** será el **cambio en W_i**.
- La elección de **ratio de aprendizaje** es muy importante.
  - Si es **demasiado pequeño**, el **método tardará mucho tiempo** en encontrar la solución óptima.
  - Si es **demasiado grande**, los **cambios en W** también serán **muy grandes** y será muy **difícil** encontrar los **parámetros** que **minimicen** la **función de coste**.
- Ratio de aprendizaje más utilizados: 0.1, 0.01

Desvanecimiento del gradiente en RNN básica

- ¿Por qué se produce el problema del **desvanecimiento del gradiente en las RNN básicas**?.
- Estas redes se caracterizan por usar la función de activación **tanh** o **sigmoide**.

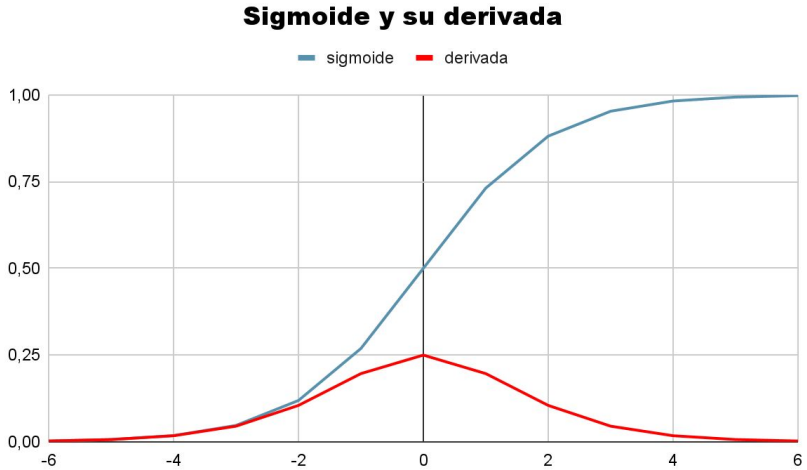
- La derivada de la **tanh** es: $tanh'(x) = 1 - tanh^2(x)$

- La función **sigmoide** y su derivada son: $\sigma(x) = \frac{1}{1 + e^{-x}}$

$$\sigma'(x) = \frac{e^{-x}}{(1 + e^{-x})^2}$$

Desvanecimiento del gradiente en RNN básica

- Los valores de las derivadas de **tanh** siempre están **entre 0 y 1**.
- **Sigmoidea** produce gradientes aún más pequeños (valores **entre 0 y 0.25**).



Desvanecimiento del gradiente en RNN básica

- En las RNN básicas, a **mayor número de capas**, durante el proceso de retropropagación (back-propagation), los **gradientes** serán **cada vez más pequeños**, y por tanto, **no se producirán cambios significativos en los parámetros**.
- Evitar el **problema del desvanecimiento del gradiente** sigue siendo un **área de investigación activa**.
- Posibles soluciones:
  - **LSTM** (Long-short Term Memory Networks)***,** también sufre el mismo problema, pero más lentamente que en RNN básicas.
  - Usar la función de activación RELU y usar normalización en los lotes.

*Hochreiter, S., Bengio, Y., Frasconi, P., & Schmidhuber, J. (2001). Gradient flow in recurrent nets: the difficulty of learning long-term dependencies.

Índice

- Redes recurrentes
- Otros ejemplos de aplicaciones de RNN en PLN
- Tipos de RNN:
  - básica (vanilla)
  - **LSTM (Long-short Term Memory Networks)**
  - GRU

Tipos de RNN: LSTM

- **LSTM** (**Long-short Term Memory Networks (LSTM)*,** se caracterizan por tener tres puertas para para decidir qué información es eliminada (**forget gate**), qué información se utiliza para modificar el estado actual (**input gate**), y qué información se pasa a la neurona siguiente (**output gate**).
- Las tres puertas utilizan distintas combinaciones de las funciones tanh y sigmoide sobre el estado anterior (C_{t-1}), el estado actual actualizado (C_t), y el dato actual a procesar (x_t).
- En este enlace puedes encontrar un tutorial que explica con más detalle la estructura de las neuronas LSTM y sus puertas: forget gate, input gate, y output gate.

Hochreiter, S., Bengio, Y., Frasconi, P., & Schmidhuber, J. (2001). Gradient flow in recurrent nets: the difficulty of learning long-term dependencies.

Tipos de RNN: LSTM

- Son capaces de detectar las dependencias en secuencias largas,
- Son más robustas (manejar el ruido o la falta de datos) en comparación a otras redes.
- Pueden ser utilizadas para una gran variedad de aplicaciones: clasificación de textos, generación de textos, reconocimiento de entidades, etc.
- No eliminan por completo el problema del desvanecimiento del gradiente (aunque son mejores que las RNN básicas).
- Su complejidad temporal es alta.
- Además, tienen un gran número de hiper-parámetros que deben ser ajustado para obtener buenos resultados.
- Necesitan ser entrenados con datasets grandes. Tienden al sobreaprendizaje en datasets pequeños.

Índice

- Redes recurrentes
- Otros ejemplos de aplicaciones de RNN en PLN
- Tipos de RNN:
  - básica (vanilla)
  - LSTM (Long-short Term Memory Networks)
  - **GRU**

Tipos de RNN: GRU

- Al igual que las LSTM, **Gated recurrent units*,** fueron propuestas para resolver el problema del **desvanecimiento del gradiente**.
- Su estructura es **más simple** que las células **LSTM**, porque sólo utilizan dos puertas: **updated gate** y **reset gate**, cuya función es decidir qué información debe pasarse a la salida, y qué información debe ser eliminada porque no es relevante para la predicción de la red.
- En este enlace, puedes encontrar un tutorial que explica con más detalle la estructura de las GRU.

Kyunghyun Cho, Bart van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. 2014. [On the Properties of Neural Machine Translation: Encoder–Decoder Approaches. In *Proceedings of SSST-8.* (https://aclanthology.org/W14-4012

Tipos de RNN: GRU

- Como son más sencillas que las LSTM, son **más eficientes.**
- **Mejor elección** que las LSTM si la **arquitectura** es **sencilla** o se cuentan con **pocos recursos** computacionales.
- Sin embargo, muestran **peores resultados** en el tratamiento de **secuencias largas**.
- También presentan más problemas de **sobreaprendizaje** que las LSTM, en especial cuando se trabaja con datasets pequeños.
- Al igual que las LSTM, para obtener **buenos resultados** es necesario **ajustar** sus hiper-**parámetros** (número de unidades, learning rate,

Resumen

- **RNN** son apropiadas para **procesar secuencias**.
- Toman **decisiones** en función del **pasado** (estados anteriores).
- Su principal **desventaja** es que **no** puede ser **paralelizada**.
- Problemas de una **RNN básica** (tanh o sigmode): **no detectan dependencias** en **secuencias largas** y **desvanecimiento del gradiente**.
- **LSTM,** tipo de RNN, pueden **capturar dependencias entre elementos distantes**.
- **GRU más** simple y **eficiente** que LSTM.

OpenCourseWare
Procesamiento de Lenguaje Natural con Aprendizaje Profundo,

Gracias!!!

https://github.com/isegura