

NLP

Redes neuronales recurrentes (RNN)

Ing. Hernán Contigiani
hernan4790@gmail.com

Programa de la materia



Clase 1: Introducción a NLP, Vectorización de documentos.

Clase 2: Preprocesamiento de texto, librerías de NLP y Rule-Based Bots.

Clase 3: Word Embeddings, CBOW y SkipGRAM, representación de oraciones.

Clase 4: Redes recurrentes (RNN), problemas de secuencia y estimación de próxima palabra.

Clase 5: Redes LSTM, análisis de sentimientos.

Clase 6: Modelos Seq2Seq, traductores y bots conversacionales.

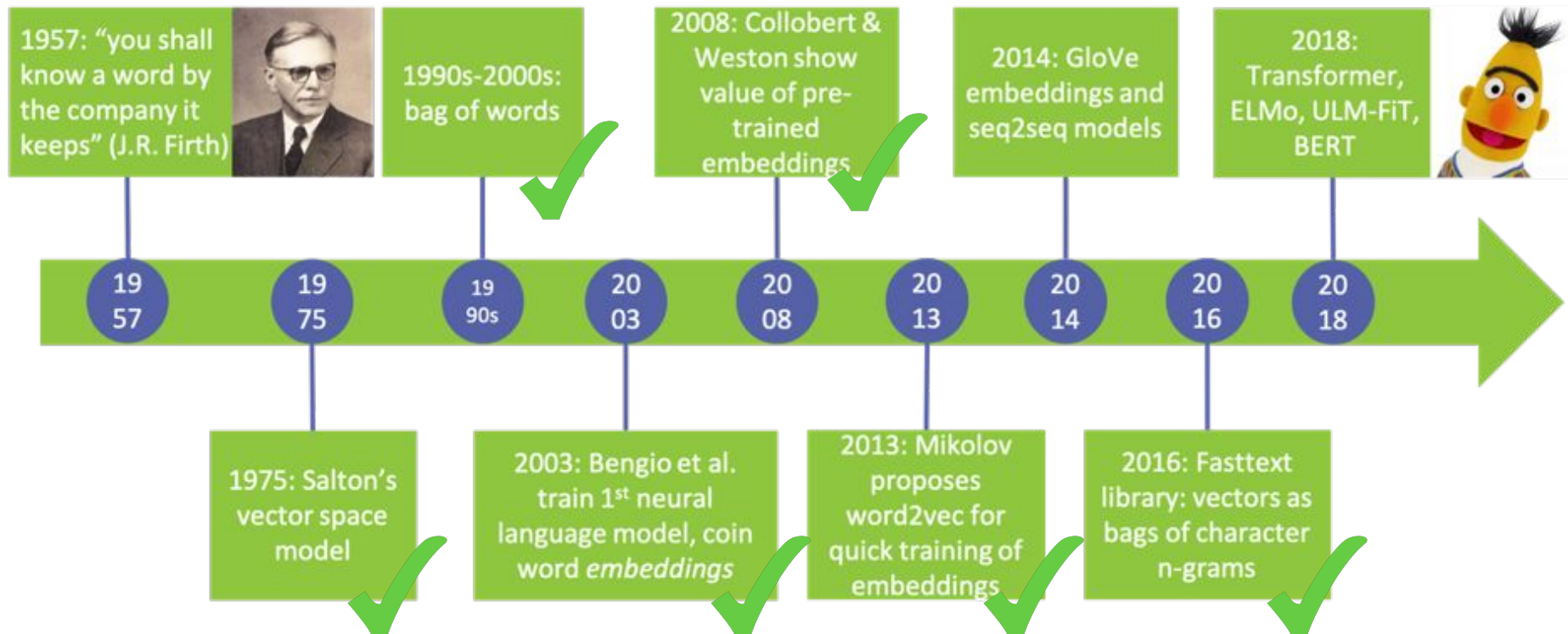
Clase 7: Celdas con Attention. Transformers, BERT & ELMo, fine tuning.

Clase 8: Cierre del curso, NLP hoy y futuro, deploy.

*Unidades con desafíos a presentar al finalizar el curso.

*Último desafío y cierre del contenido práctico del curso.

Timeline



Redes Neuronales Recurrentes (RNN)



"Se introduce este tipo de celda neuronal para lograr que información del pasado influya en los resultados futuros".



Se utiliza principalmente para resolver problemas de secuencia, en donde el valor anterior está relacionado con el valor futuro.



Se busca que el modelo aprenda a generalizar y compartir features entre palabras en diferentes posiciones.

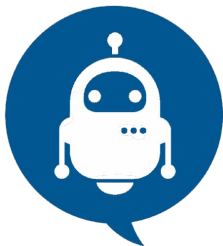


Permite construir modelos cuyos vectores de entrada o salida no posean una dimensión fija.

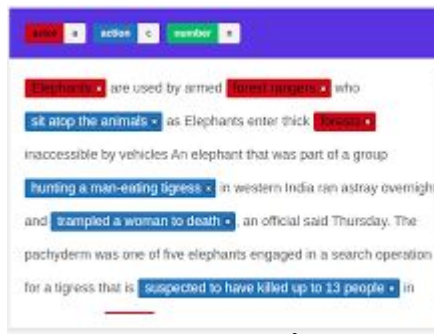
$$\prod_{i=1}^{i=m} P(w_i | w_{i-(n-1)}, \dots, w_{i-1})$$

*"Hoy el **día** está **hermoso** y **despejado**, se puede ver un hermoso **cielo... azul**"*

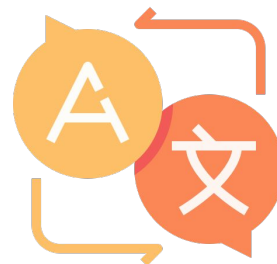
Soluciones dónde la secuencia es importante



Bots
Conversacionales



Name entity
recognition



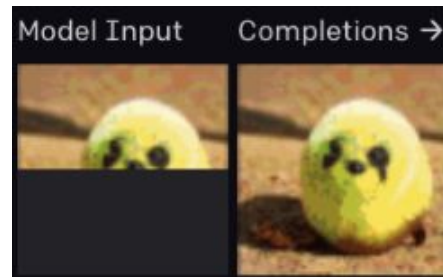
Traducción de
idiomas



Speech to text



Generar música

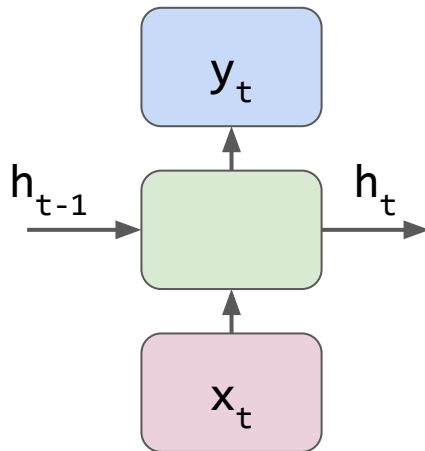


Completar una
imagen

Celda RNN

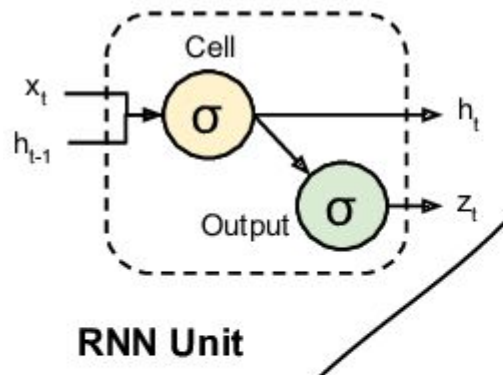


[LINK](#)

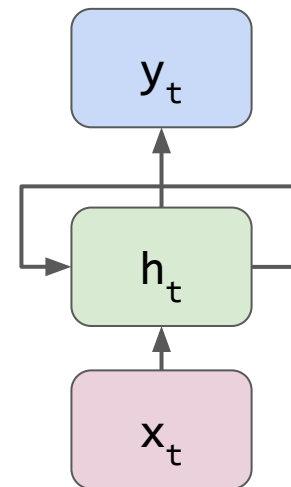


Unidad básica

$$h_t = \sigma(W_{hh} * h_{t-1} + W_{hx} * x)$$
$$y_t = \text{softmax}(U[h_t])$$

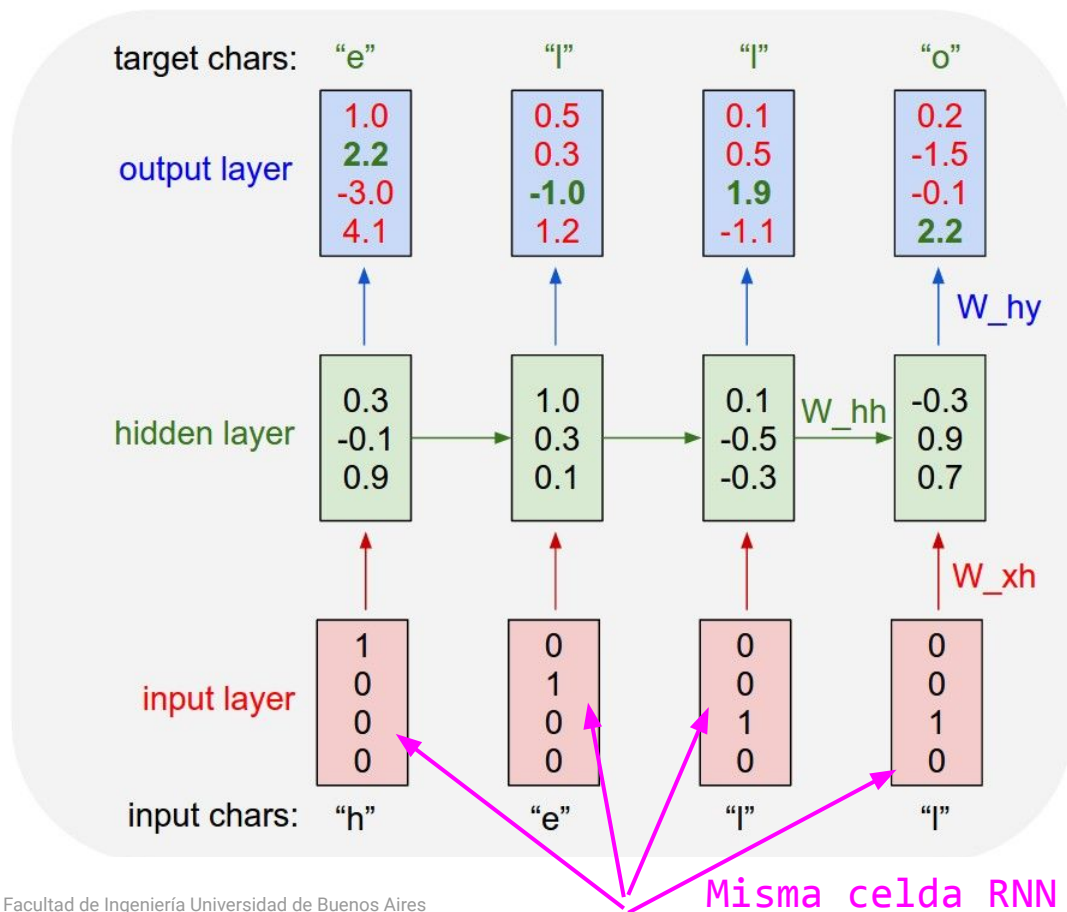


forward



Representación
compacta

Propagación (ejemplo)

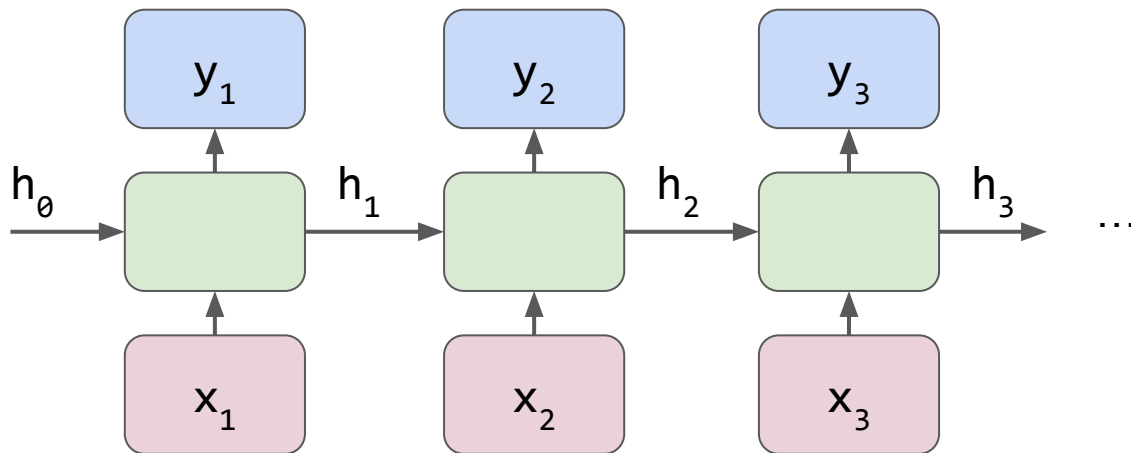


En este ejemplo entra una palabra/letra y sale otra

En estas redes de secuencia su grafo de cómputo es en serie, no es posible paralelizar, ya que el estado futuro depende del estado anterior.

Con cada salida se actualizan los pesos W_{hh} , W_{xh} y W_{hy} para el próximo cómputo

Implementación básica (una sola celda)



```
rnn = RNN()
y1 = rnn.step(X1)
y2 = rnn.step(X2)
y3 = rnn.step(X3)
```

h_0 inicia en cero o random

```
class RNN:
```

```
# ...
```

```
def step(self, x):
```

```
    # update the hidden state
```

```
    self.h = np.tanh(np.dot(self.W_hh, self.h) + np.dot(self.W_xh, x))
```

```
    # compute the output vector
```

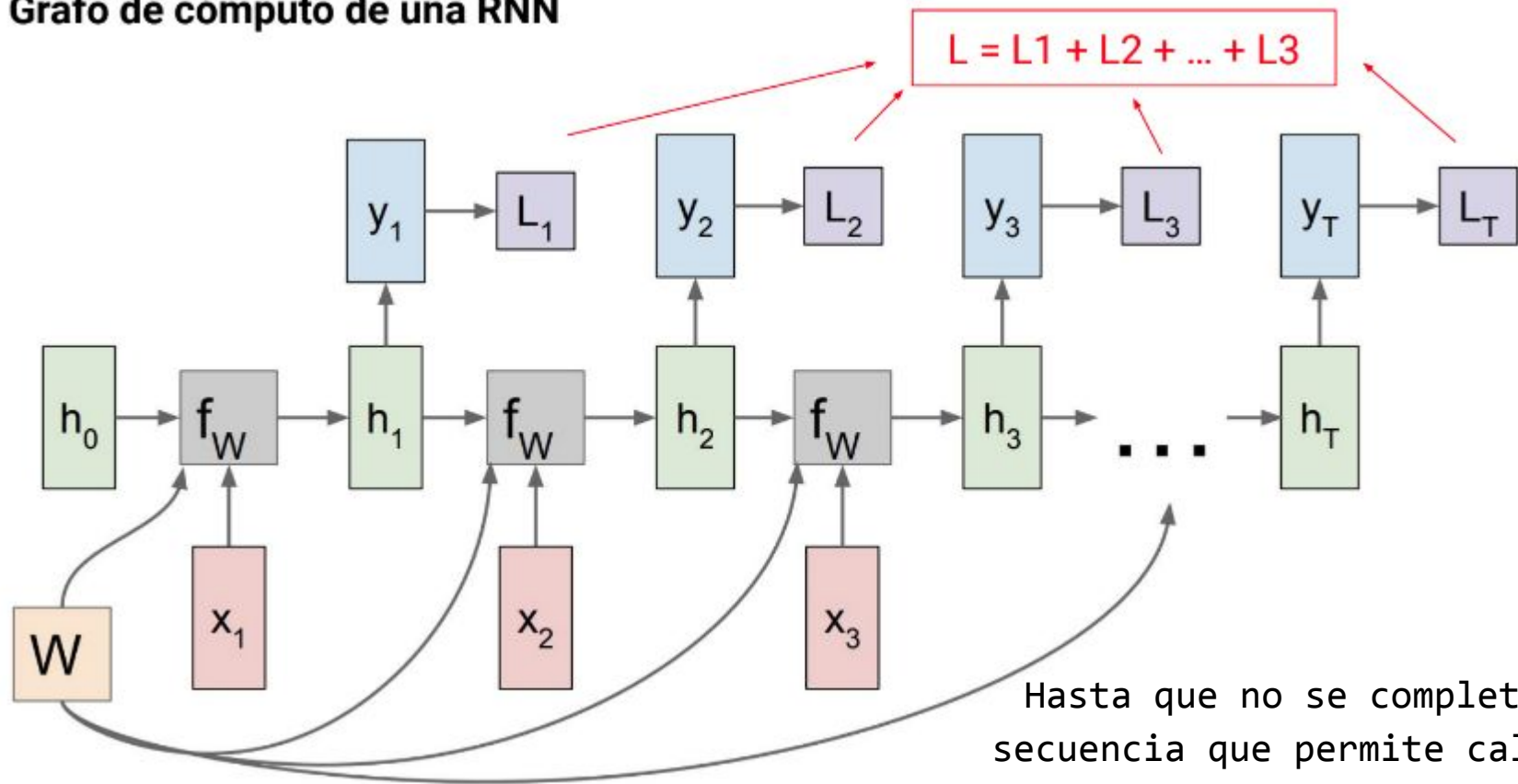
```
    y = np.dot(self.W_hy, self.h)
```

```
    return y
```

$$h_t = \sigma(W_{hh} * h_{t-1} + W_{hx} * x)$$
$$y_t = \text{softmax}(U[h_t])$$

[LINK](#)

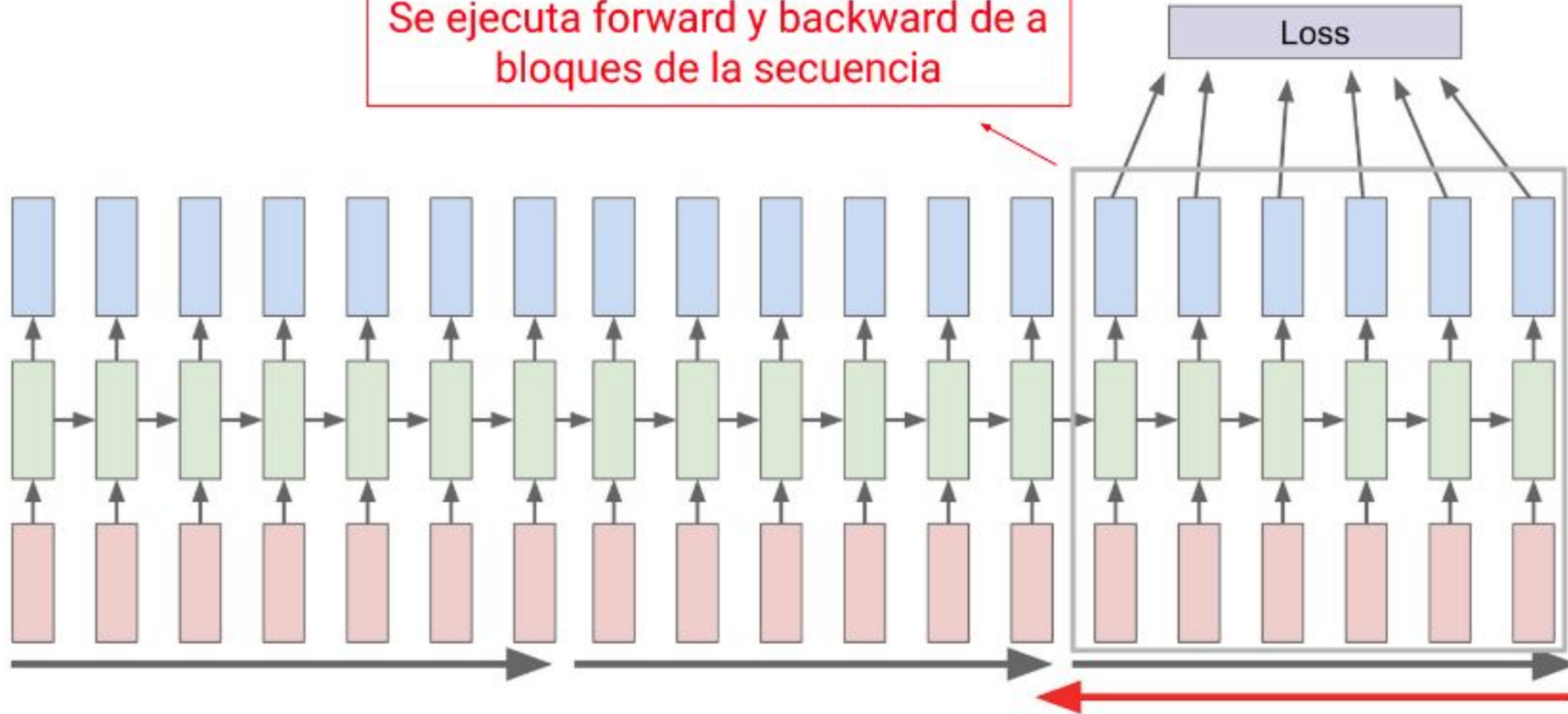
Grafo de cómputo de una RNN

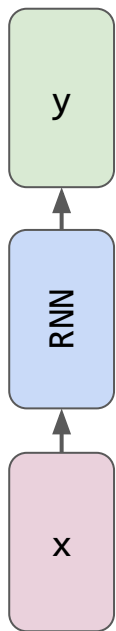


Hasta que no se complete la secuencia que permite calcular el loss no se actualizan los pesos (W) de la/s celda/s

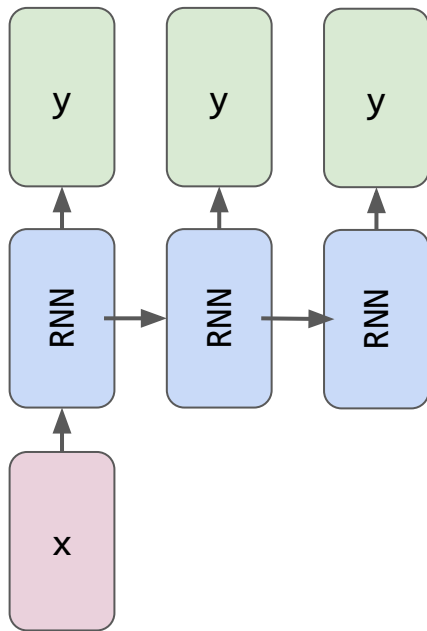


Se ejecuta forward y backward de bloques de la secuencia

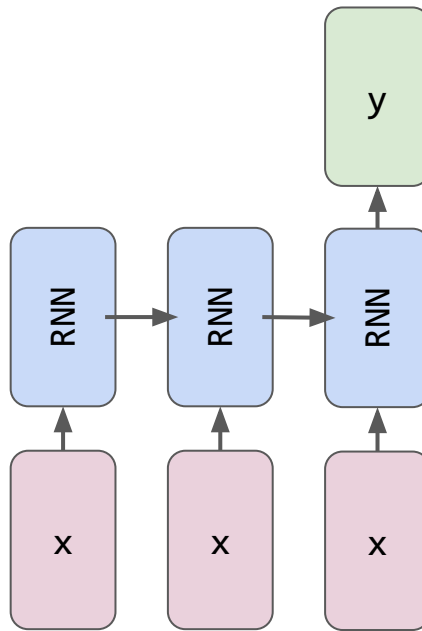




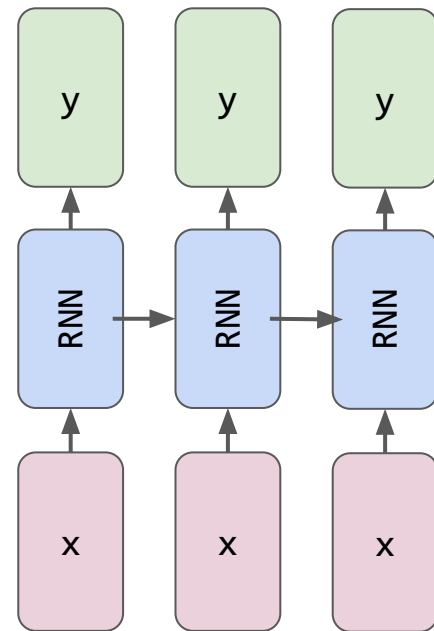
one-to-one



one-to-many



many-to-one

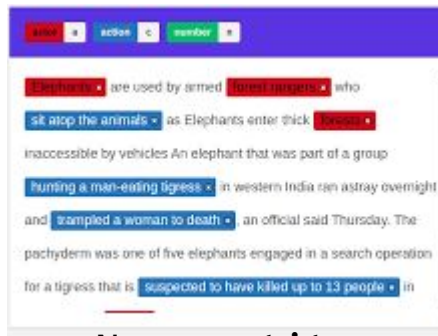
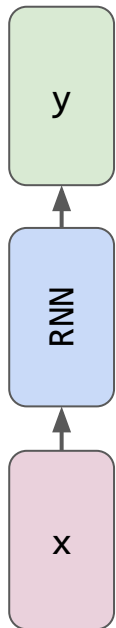


many-to-many

one-to-one



"Por cada entrada al sistema (word2vec) la celda arroja un resultado"



Name entity
recognition

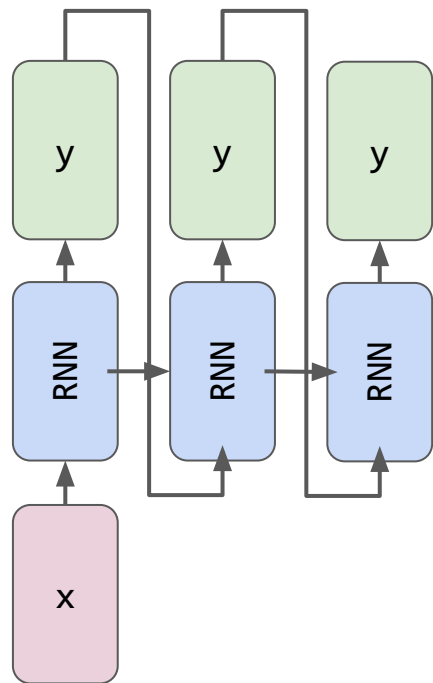


Se utiliza principalmente para reconocer las entidades principales en una oración o frase (encontrar el sujeto de la oración o palabras importantes).

one-to-many



"Dada una entrada de tamaño fijo el sistema arroja una sentencia o oración a partir de ella de longitud variable"



Este tipo de estructuras se utilizan para generar una secuencia a partir de un punto de partida. Se podría a partir de una sílaba completar una palabra o partir de una nota armar un ritmo de música.

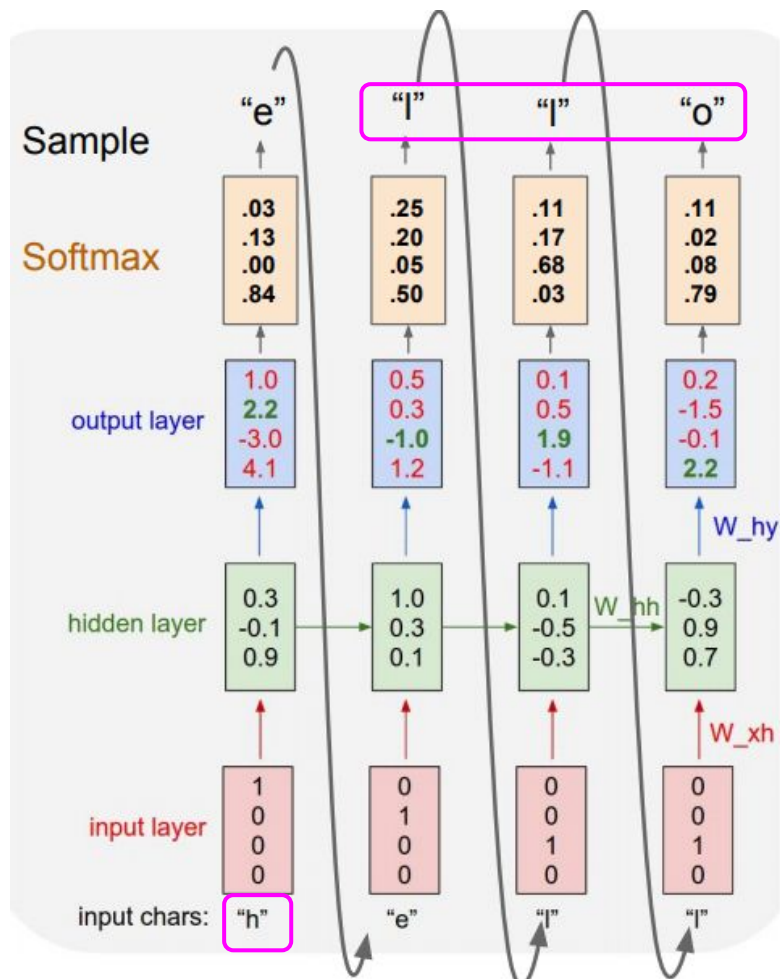


Generar música



Autocompletar

one-to-many





Link al Colab



LINK



Link al Colab

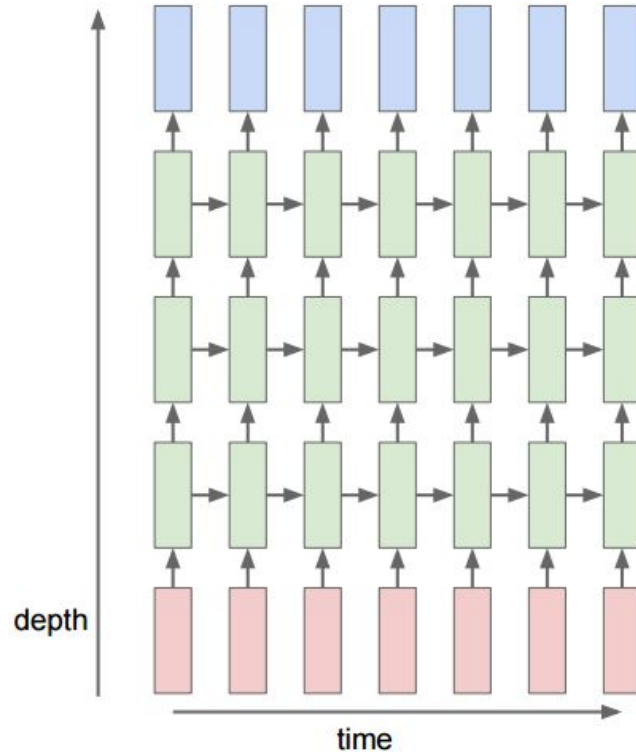


LINK

Multi-layer RNN



Tal como se vio en los ejemplos se trata de apilar layers RNN en donde la salida de una se traslada a la entrada de la siguiente



Problema de una RNN tradicional

[LINK](#)



"Una RNN tradicional solo usa información del pasado y no de las futuras palabras para predecir"

Ejemplo: Data una sentencia determinar si existe una entidad que represente al nombre de una persona utilizando (name entity recognition)

Ejemplo 1:

*"Hoy escuche que **Victoria** terminó su bot para NLP" → persona*

Ejemplo 2:

*"Hoy escuche que **Victoria** cambió de intendente" → ciudad*

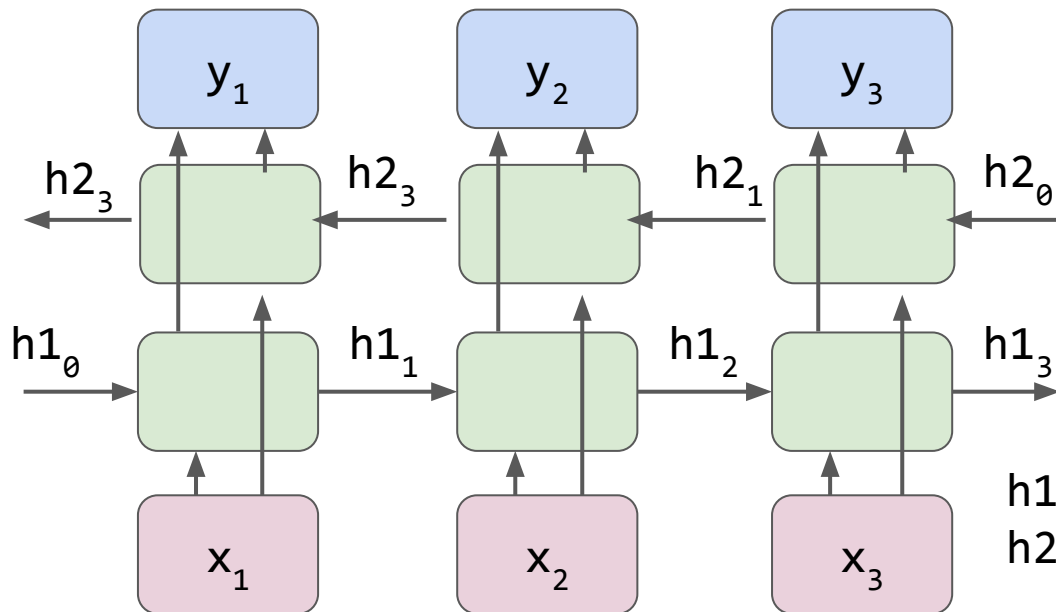
La
contextualización
de la palabra es
a futuro

Bidirectional RNN (BRNN)

BRNN PAPER



“La palabra anterior y la palabra futura tienen impacto en la presente predicción”



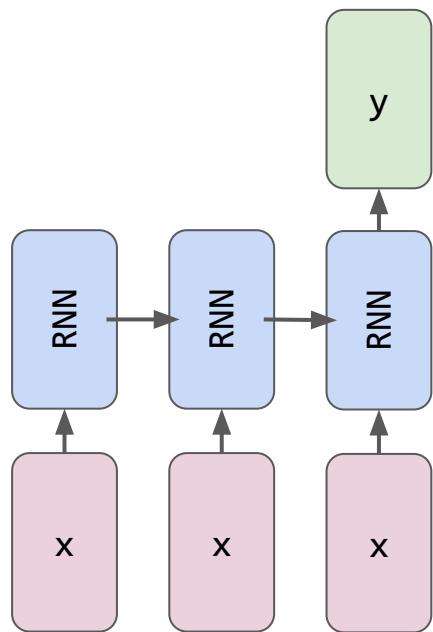
$$h1_t = \sigma(W_{hh1} * h_{t-1} + W_{hx1} * x)$$
$$h2_t = \sigma(W_{hh2} * h_{t+1} + W_{hx2} * x)$$

$$y_t = \text{softmax}(U[h1_t + h2_t])$$

many-to-one



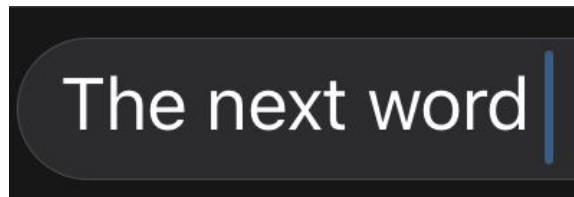
"Dada una sentencia o oración de entrada de tamaño fijo, el sistema arroja un único resultado que la representa".



many-to-one



Este tipo de estructuras se utilizan para determinar cuál es la siguiente palabra o elemento en la secuencia o para clasificación (sentiment analysis).



Predicción de próxima palabra



Análisis de sentimientos



Link al Colab



LINK

Text prediction



Se utilizará many-to-one, por lo que hay que seleccionar la dimensión de la sentencia de entrada y dividir el texto en grupos:

The next word

Predicción de
próxima palabra

Sentencia

'Yesterday, all my troubles seemed so far away'

Tokens

['yesterday', 'all', 'my', 'troubles', 'seemed', 'so', 'far', 'away']

Vectores de entrada de 4 tokens

```
[['yesterday', 'all', 'my', 'troubles'],  
 ['all', 'my', 'troubles', 'seemed'],  
 ['my', 'troubles', 'seemed', 'so'],  
 ['troubles', 'seemed', 'so', 'far']]
```



Link al Colab



LINK



Utilizar otro dataset
y poner en práctica
la predicción de
próxima palabra





¡Muchas gracias!