

Redes Neuronales Recurrentes

Prof. Wílmer Pereira

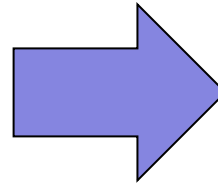
Análisis de texto

La comprensión del lenguaje natural es un tema mítico en IA que ha visto varios fracasos. Una nueva luz se perfila con las redes neuronales profundas aunque, por ahora, para dominios específicos con vocabulario y objetivos bien delimitados ...

- En la década de los sesenta, en plena guerra fría, se pretendió realizar un traductor automático de ruso a inglés. Todo el esfuerzo y dinero terminó en resultados decepcionantes. Otro intento fue el proyecto japonés de computadoras de 5^{ta} generación cuyo lenguaje ensamblador estaba basado en PROLOG ... se canceló sin avances ...

- El reconocimiento de lenguaje natural requiere considerar al menos 5 fases:

- | | |
|---------------|------------------|
| 1. Fonética | 2. Lexicográfica |
| 3. Sintáctica | 4. Semántica |
| 5. Pragmática | |



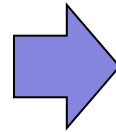
El esfuerzo conceptual y de cálculo es enorme, con dificultades para dar respuesta en tiempos razonables

- A pesar de contar con las redes neuronales profundas, actualmente sólo podemos aspirar a problemas de lenguaje natural reducidos como análisis de sentimientos, clasificación de documentos, identificación de autor (música o texto), responder preguntas (en contexto bien restringidos). Por supuesto, sin verdadero entendimiento del discurso, más bien con correspondencias, mediante estructuras estadísticas, del lenguaje escrito.

Deep learning y análisis de texto

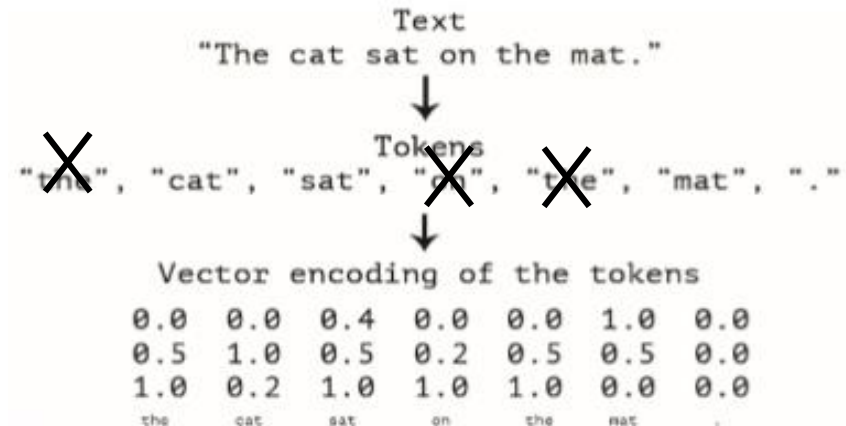
- Como punto de partida, se debe decidir como dividir el texto en unidades atómicas, a nivel sintáctico, en **n-gramas** (secuencias de palabras). Esto permite darle contexto a las palabras y resolver la polisemia (banco, bolsa, botín, botón, capital, ...)

El gato sube despacio la escalera
Con el gato se sube despacio el coche



Con 4-gramas, se resuelve el significado semántico de gato

- Por otro lado, la idea es asociar más de un número a cada n-grama, es decir, un vector. Esto se logra mediante el proceso de *word embedding*.



- Lo más común son n-gramas de una palabra o *tokens* ... Estas se almacenan en un repositorio, que se comporta como un conjunto, conocido como **bag-of-words**.
- Además, en algunas circunstancias, puede ser conveniente eliminar las palabras que no aportan significado semántico: artículos, preposiciones, conjunciones, ... Este filtrado se realiza gracias a los **stop words** que dependen de cada idioma. En ocasiones no se considera este filtrado ...

Palabras en el bag-of-word

- El *bag-of-word* puede ser muy grande por lo que para disminuirlo se agrupan varias palabras bajo un representante. Por ejemplo, se podrían eliminar sufijos, prefijos, conjugaciones de verbos, etc ... Para ello se propone almacenar sólo un *token* por palabras relacionadas. Este proceso se puede realizar de dos maneras: ***Lemmatization*** y ***Stemming***.

Lemmatization

- La *lemmatization* es el proceso donde dada la forma flexionada, retorna la palabra que etimológicamente la representa. Por ejemplo, para la conjugación de un verbo se usa el infinitivo o para un adjetivo el masculino singular, ... En consecuencia, la palabra que se incluye en el *bag-of-word* pertenece al lenguaje. Así el lema de canto, canté o cantaríamos, ... sería cantar.
- Hay dos maneras de lematizar: morfológicamente y sintácticamente. Visto morfológicamente la palabra ama tendría dos lemas: el sustantivo ama y el verbo amar. En cambio sintácticamente se puede diferenciar identificando la estructura gramatical de la frase. Por ejemplo, en la frase: “La ama de llaves abrió la puerta”, el lema es ama y no amar ...

Stemming

- El *stemming*, por el contrario, no asegura que la palabra del *bag-of-word* pertenezca al lenguaje porque recuperar la raíz común a todas sus declinaciones. Por ejemplo, para canto, canté o cantaríamos sería el *token* cant ... Otra diferencia, con respecto a la *lemmatization*, es que el *token* se obtiene a partir de un proceso lexicográfico ...
- También es común utilizar el TF-IDF para calcular la frecuencia de aparición de los lemas en documento y compensada por la frecuencia de la palabra en la colección de documentos. Esto ayuda a determinar la relevancia y a construir el *stop word* con los *tokens* o lemas menos importante y el *bag-of-words* con los más importantes ...

TF-IDF

- La idea es bajar el peso de las palabras que se usan muy frecuentemente y aumentar el peso de las palabras poco comunes si están en sólo un documento:

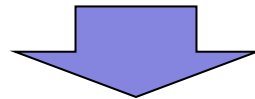
$$TF(\text{término}, \text{documento}) = \frac{n_j}{\sum_{k=1}^w n_k} \quad IDF(\text{término}) = \log \frac{N}{n_t}$$

$$TF - IDF(\text{término}, \text{documento}) = TF(\text{término}, \text{documento}) \times IDF(\text{documento})$$

One-hot encoding

- Es un proceso que construye una matriz binaria dispersa, transformando cada valor categórico de un atributo en una columna de la matriz.

Label Encoding			One Hot Encoding			
Food Name	Categorical #	Calories	Apple	Chicken	Broccoli	Calories
Apple	1	95	1	0	0	95
Chicken	2	231	0	1	0	231
Broccoli	3	50	0	0	1	50



Eliminar atributos categóricos para poder procesar numéricamente

- En Keras se puede realizar muy rápidamente gracias a la clase `Tokenizer`.

```
from keras.preprocessing.text import Tokenizer

samples = ['The cat sat on the mat.', 'The dog ate my homework.']

# We create a tokenizer, configured to only take
# into account the top-1000 most common on words
tokenizer = Tokenizer(num_words=1000)
# This builds the word index
tokenizer.fit_on_texts(samples)

# This turns strings into lists of integer indices.
sequences = tokenizer.texts_to_sequences(samples)

# You could also directly get the one-hot binary representations.
# Note that other vectorization modes than one-hot encoding are supported!
one_hot_results = tokenizer.texts_to_matrix(samples, mode='binary')

# This is how you can recover the word index that was computed
word_index = tokenizer.word_index
print('Found %s unique tokens.' % len(word_index))
```

Ejemplo One-hot encoding

Sean, por ejemplo, los siguientes tres textos

El gato se sienta en su alfombra

El perro come su alimento

El perro y el gato son animales domésticos

La matriz dispersa sería

	alimento	alfombra	animales	come	domestico	el	en	gato	...	perro	se	sienta	son	su	y	...
Texto 1		1				1	1	1			1	1		1		
Texto 2				1		1				1				1		
Texto 3	1		1		1	1		1		1			1		1	

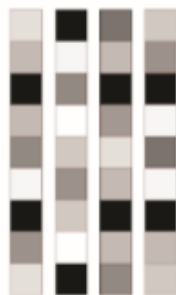
Es de notar que las palabras que se repiten más de una vez se reportan sólo una vez por fila o reglón

Word embeddings



One-hot word vectors:

- Sparse
- High-dimensional
- Hard-coded



Word embeddings:

- Dense
- Lower-dimensional
- Learned from data

En lugar de asociar una matriz binaria dispersa, que desperdicia espacio, se define un vector de gran dimensionalidad, generalmente son entre 100 a 1000 valores ... baja con respecto a *one-hot-encoding* ...

Hay dos maneras de obtener los vectores por palabra:

1. Usar los textos de entrada para obtener los vectores de cada palabra mediante entrenamiento con una red neuronal.
2. Cargar un espacio de palabras con los vectores ya definidos, pre-entrenado con grandes bases de datos de palabras.

- Espacios n -dimensionales con vectores de palabras pre-entrenada se usan para lograr transferencia de aprendizaje. Hay dos repositorios muy conocidos:

Word2Vec

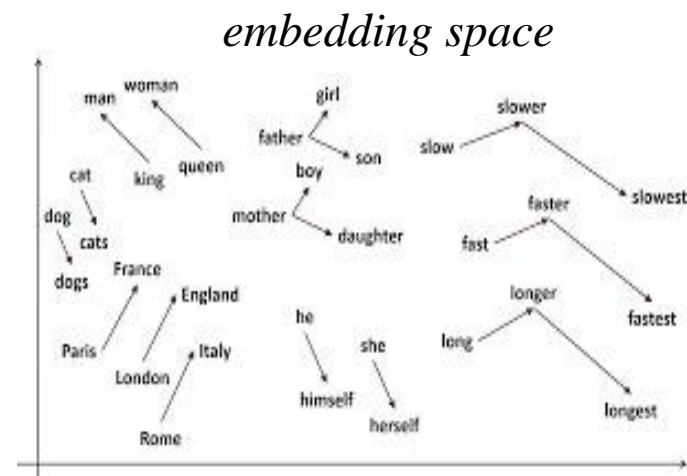
Desarrollado por Google en 2013 (tiene cientos de billones de palabras). Cada palabra es un vector o tensor 1D de 300 valores obtenidos del proyecto [políglota](#) ...

GloVe

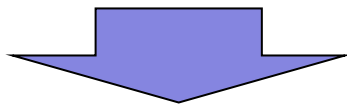
Proyecto de Stanford (2014) con funcionalidades similares a Word2Vec.

Intuición de word embedding

- Cada palabra se sitúa en un espacio n -dimensional y se agrupan por similaridad. Además dado un vector de transformación se pueden acercar conceptos
- ¿Existe un *embedding space* para cualquier tipo de discurso a procesar? Quizás sea posible pero sólo se ha logrado para dominios específicos. Uno de los problemas es la polisemia ... [Animación](#) ...



- Keras ofrece una capa para aprender *embedding space* ... la clase se invoca como `Embedding(MAXWORDS, DIMVECTOR)` ... inicialmente define un diccionario interno de palabras con vectores de tamaño `DIMVECTOR` partiendo con valores aleatorios

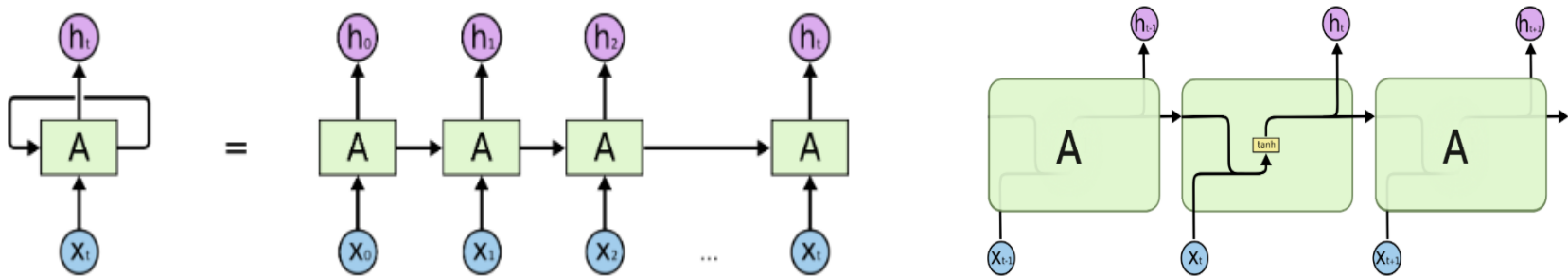


Embedding toma un vector 2D (*muestras, indicesPalabra*) como entrada y devuelve un vector 3D (*muestras, indicesPalabra, DIMVECTOR*) ... Se entrena de dos maneras: con sólo las palabras de los textos de entrada o usando un espacio de palabras pre-entrenado más las palabras del texto (Word2Vec o GloVe).

- Después de este proceso el vector 3D sirve de entrada para una red neuronal recurrente o una red neuronal convolucional 1D.

Redes neuronales recurrentes

Las mayoría de redes neuronales son *feedforward*. Al incluir lazos de retroalimentación, se recupera el valor de la salida para mantener el estado de la información pasada. Esto es imprescindible cuando se procesan series de tiempo (finanzas, meteorología, ...)



- Estas redes (RNN) son ideales en el procesamiento de texto porque el contexto es relevantes.

... pero ...

... tienen problemas de desvanecimiento de gradiente ...

- Como la dependencia a largo plazo se diluye, se han desarrollado una RNN's que mantienen por más tiempo el valor de los estados precedentes ... La primera versión conocida como LSTM (*Long Short Term Memory Network*) a veces mantienen el estado de largo plazo y a veces lo olvidan para ser substituido por el estado más reciente ... Fue propuesta por Hochreiter&Schmidhuber (1997) ... El estado del arte son los *Transformer*.

RNN con Keras

- Existen muchos tipos de redes neuronales recurrentes. La más simple, conocida como SimpleRNNLayer, puede configurarse para que retorne una sólo salida al final de la recurrencia o una salida por cada serie de tiempo o entrada

```
>>> from keras.models import Sequential
>>> from keras.layers import Embedding, SimpleRNN
>>> model = Sequential()
>>> model.add(Embedding(10000, 32))
>>> model.add(SimpleRNN(32))
>>> model.summary()
```

Layer (type)	Output Shape	Param #
embedding_22 (Embedding)	(None, None, 32)	320000
simplernn_10 (SimpleRNN)	(None, 32)	2080

=====
Total params: 322,080
Trainable params: 322,080
Non-trainable params: 0

```
>>> model = Sequential()
>>> model.add(Embedding(10000, 32))
>>> model.add(SimpleRNN(32, return_sequences=True))
>>> model.summary()
```

Layer (type)	Output Shape	Param #
embedding_23 (Embedding)	(None, None, 32)	320000
simplernn_11 (SimpleRNN)	(None, None, 32)	2080

=====
Total params: 322,080
Trainable params: 322,080
Non-trainable params: 0

```
>>> model = Sequential()
>>> model.add(Embedding(10000, 32))
>>> model.add(SimpleRNN(32, return_sequences=True))
>>> model.add(SimpleRNN(32, return_sequences=True))
>>> model.add(SimpleRNN(32, return_sequences=True))
>>> model.add(SimpleRNN(32)) # This last layer only returns the last outputs.
>>> model.summary()
```

Layer (type)	Output Shape	Param #
embedding_24 (Embedding)	(None, None, 32)	320000
simplernn_12 (SimpleRNN)	(None, None, 32)	2080
simplernn_13 (SimpleRNN)	(None, None, 32)	2080
simplernn_14 (SimpleRNN)	(None, None, 32)	2080
simplernn_15 (SimpleRNN)	(None, 32)	2080

=====
Total params: 328,320
Trainable params: 328,320
Non-trainable params: 0

También son posible varias capas recurrentes consecutivas

IMBD (reseñas de cine) con RNN

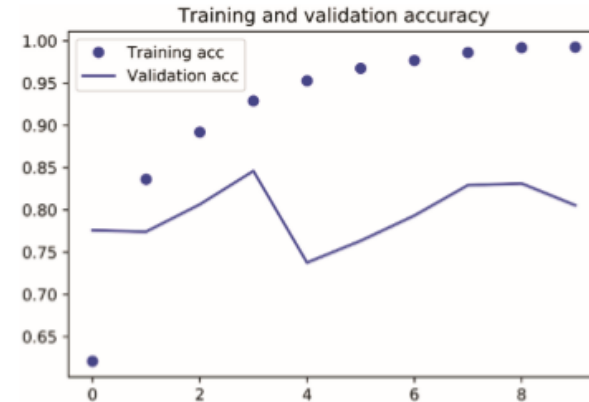
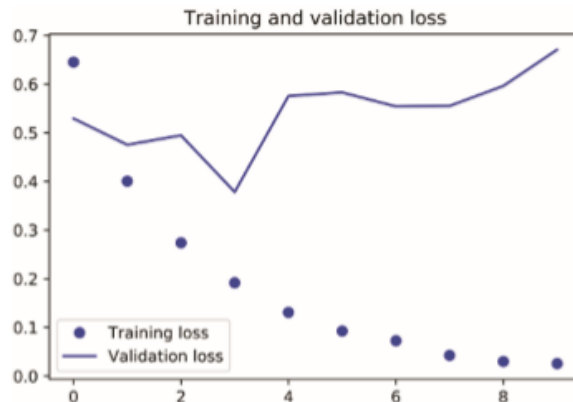
- Con redes neuronales convencionales (16/16/1) se obtuvo un *accuracy* del 88% ... veamos si mejora con las RNN ...

```
from keras.layers import Dense

model = Sequential()
model.add(Embedding(max_features, 32))
model.add(SimpleRNN(32))
model.add(Dense(1, activation='sigmoid'))

model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc'])
history = model.fit(input_train, y_train,
                    epochs=10,
                    batch_size=128,
                    validation_split=0.2)
```

Si se muestra el comportamiento durante el entrenamiento y se calcula el *accuracy* con los datos de prueba



- Ahora con una RNN (32/1) se obtuvo un *accuracy* del 85% y desafortunadamente con *overfitting* ... las razones pueden ser por tratarse de una red neuronal más pequeña y/o, la más probable, que este tipo de RNN simple no se adecuaba a texto ...

RNN simple y CNN+RNN

- En primer lugar una [explicación alternativa](#) en inglés de las RNN (1 hora ...)
- Un buen ejemplo para RNN son las series de tiempo de las manchas solares ... En este [video](#) (13 minutos en inglés) se muestra la programación básica en Keras y luego el ejemplo de las manchas solares
- Una aplicación para video con CNN+RNN en Keras en un [video](#) (22 minutos en inglés)
- Una alternativa más rápida para las RNN las redes neuronales transformacionales en el siguiente [video](#) (13 minutos en inglés). No tienen ejemplos de código sólo teoría ...
- En *Deep Learning* están aún las redes neuronales generativas como son las *Variational Autoencoders* y las GAN (*Generative Adversarial Network*) que también son de gran interés ...