Redes Neuronales Recurrentes

Prof. Wílmer Pereira

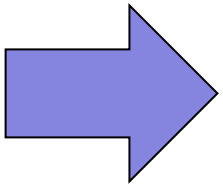
Análisis de texto

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 japones de computadoras de 5ta 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.

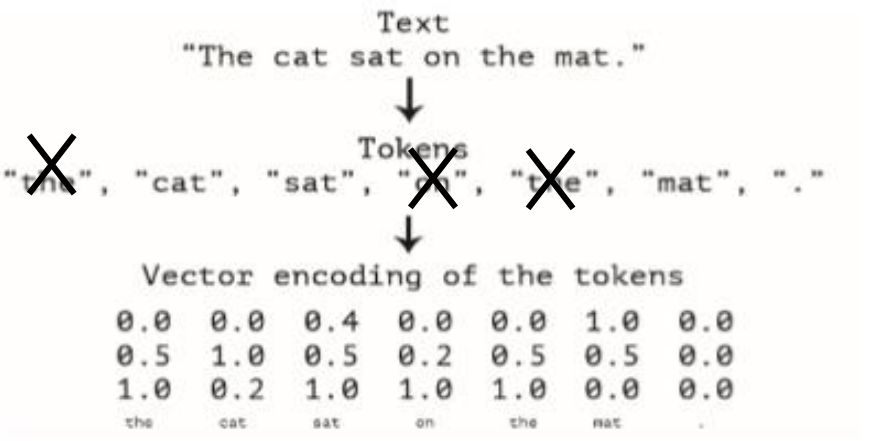
Deep learning y análisis de texto

- Como punto de partida, se debe vectorizar el texto transformándolo en tensores. Los insumos son palabras aisladas o **n-gramas** (secuencias de palabras) para darle contexto a las palabras y resolver la polisemia (banco, bolsa, botín, botón, cabo, 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

- En consecuencia, el texto debe estar dividido en palabras o n-gramas que llamaremos *tokens*. La idea es asociar más de un número a cada *token*, es decir, un vector (tensor 1D). Esto se logra mediante el proceso de *word embedding*.



- De ahora en adelante sólo consideraremos palabras 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

- Además para reducir el tamaño del *bag-of-word*, se podrían eliminar, por ejemplo, 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

- TF significa frecuencia del término e IDF frecuencia del inversa del documento. La idea es bajar el peso de las palabras que se usan muy frecuentemente y aumentar el peso de las palabras poco comunes que ocurren solamente en ese documento:

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

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

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

Food Name	Categorical #	Calories
Apple	1	95
Chicken	2	231
Broccoli	3	50

→

One Hot Encoding

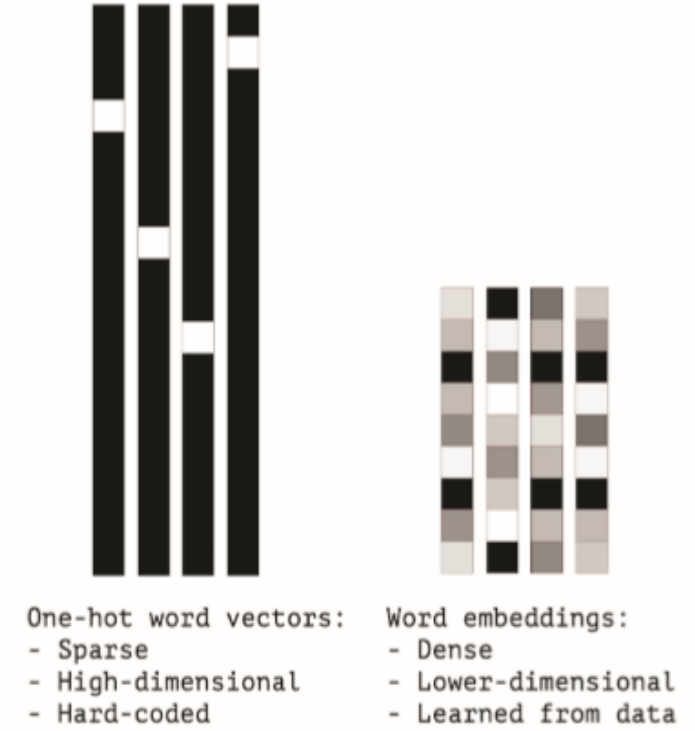
Apple	Chicken	Broccoli	Calories
1	0	0	95
0	1	0	231
0	0	1	50

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))
```

Word embeddings



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.

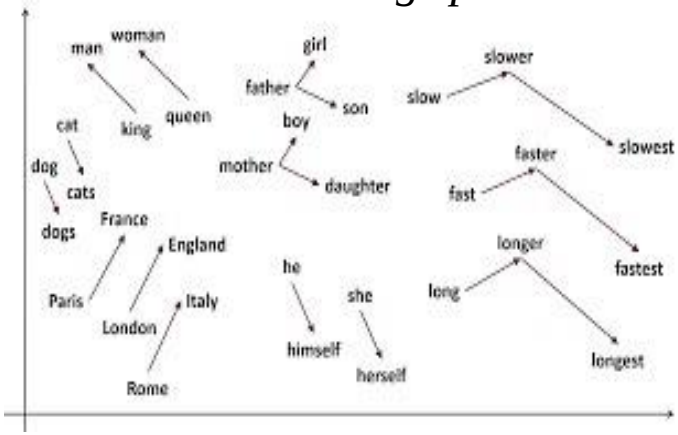
- Los dos espacios *n*-dimensionales con vectores de palabras pre-entrenada para lograr transferencia de aprendizaje. Los repositorios más utilizados son:

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

embedding space



- ¿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 …

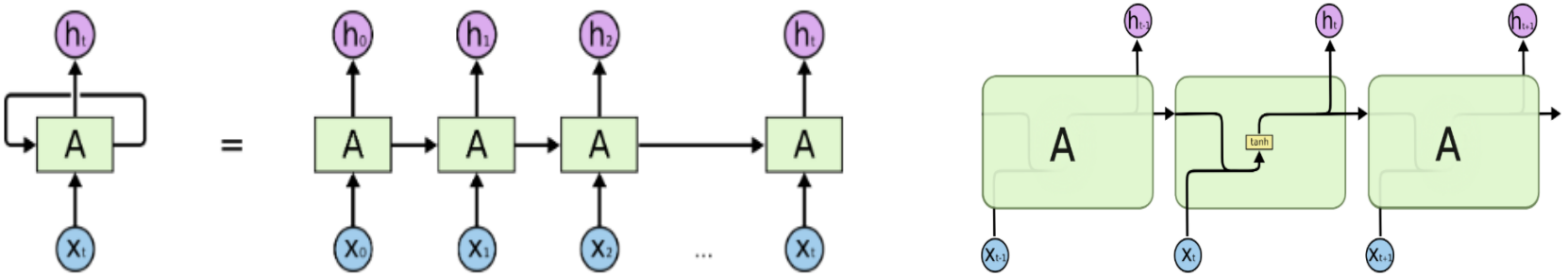
- Keras ofrece una capa para aprender *embedding space* … la clase se invoca como `Embedding(`*`MAXWORDS`*`,`*`DIMVECTOR`*`)` … inicialmente define un diccionario interno de palabras con vectores de tamaño *`DIMVECTOR`* partiendo con valores aleatorios

`Embedding` toma un tensor 2D (*muestras*, *indicesPalabra*) como entrada y devuelve un tensor 3D (*muestras*, *indicesPalabra*, *`DIMVECTOR`*) … Se entrena con sólo las palabras de los textos de entrada o usando un espacio de palabras pre-entrenado (Word2Vec o GloVE).

- Después de este proceso el tensor 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, series de tiempo, …

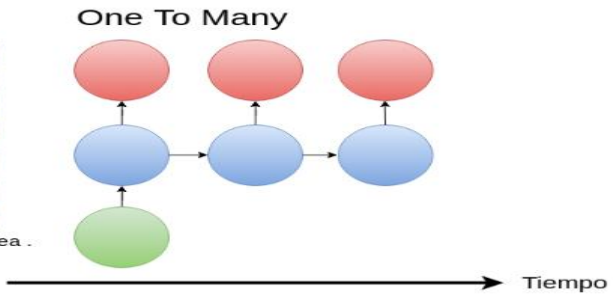
  … pero …

  … tienen problemas de desvanecimiento de gradiente …

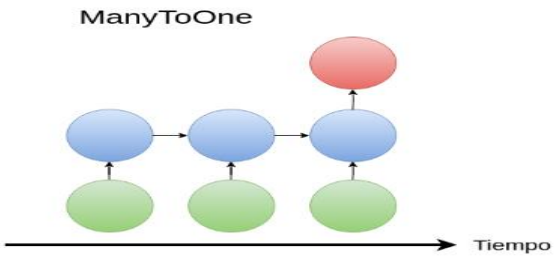
- Como la dependencia a largo plazo se diluye, se desarrolló una RNN que mantiene por más tiempo el valor de los estados precedentes … Estas redes se conocen como LSTM (*Long Short Term Memory Network*) a veces mantienen el estado de largo plazo y a veces lo olvidan para ser substituido por el estado más reciente … Fue propuesta por Hochreiter&Schmidhuber (1997).

Arquitecturas RNN

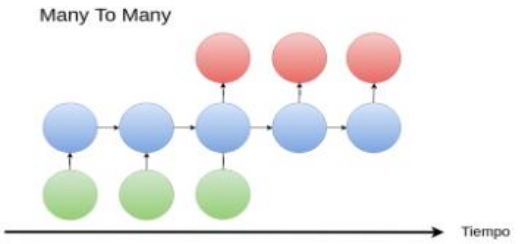
a group of giraffes standing in a grassy area .
Fuente: https://www.analyticsvidhya.com



Dada una imagen obtener un texto que la describa …



Dado una secuencia de texto clasificarlos como en el aná-lisis de sentimiento



Traducción automática entre diferentes idiomas …

Video sencillo sobre RNN y otro video más

RNN con Keras

- Existen muchos tipos de redes neuronales recurrentes. La más simple, conocida como `SimpleRNNLayer`, puede configurarse para que retorne una sóla 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
```

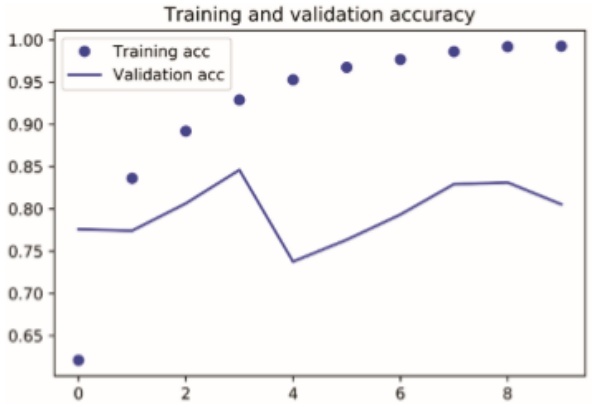
También son posible varias capas recurrentes consecutivas

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 adecúa 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 …