Jerónimo Molina Molina

Procesamiento del Lenguaje Natural

2. Word Embeddings

Operaciones con palabras

Tras un recorrido histórico, en el que hemos podido practicar con las redes neuronales más básicas en nuestra materia, hemos abordado diferentes tareas en la implementación de soluciones de procesamiento del lenguaje natural.

- Limpieza básica de un texto
- Tokenización por caracteres o palabras
- Clasificación numérica de los tokens para que los pueda procesar una red neuronal

y… si te fijas en un detalle, de este último punto, El número asignado a cada palabra en el último ejercicio del tema 1, no aportaba significado a la palabra, era, solamente, un número identificativo, ordinal si quieres, pero nada más que eso.

¿Y si pudiéramos asignar una numeración a cualquier palabra de modo que esa numeración defina su significado? ¿Y si la numeración permitiese definir su significado en diferentes contextos?

¿Podríamos realizar operaciones matemáticas con las palabras?

Te pongo un ejemplo:

Rey - Hombre + Mujer = Reina

Esto, sin duda, es otro nivel, … nos sumergimos en el mundo de los Word Embeddings, en los orígenes de la era moderna del procesamiento del lenguaje natural (y de mucho más...)

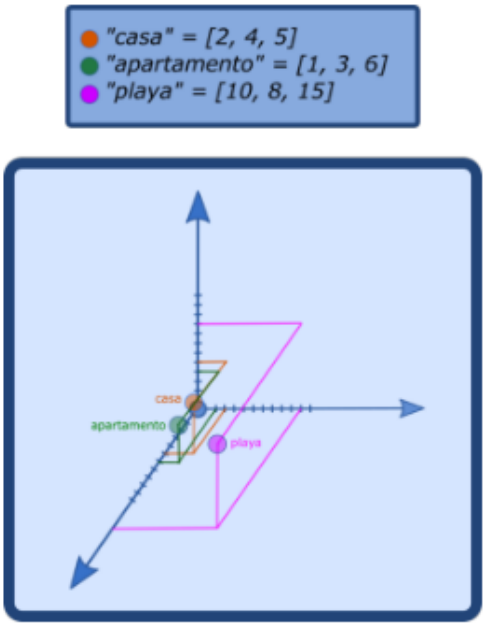
Bienvenido.

Los embeddings

Las técnicas actuales de procesamiento del lenguaje natural pasan, de forma inevitable, por la vectorización de tokens.

Evidentemente con un único número asociado a una palabra, no se puede representar su significado, pero la cosa cambia si hablamos de un vector de números, todo es diferente cuando pensamos en una representación en un espacio n-dimensional de una palabra.

Veamos el siguiente ejemplo…



En la figura se aprecian vectores descriptores, de dimensión 3, para tres palabras de ejemplo.

Los vectores describen las características de la palabra a la que representan, y además pueden situarse en un espacio tridimensional (por ser de dimensión 3).
Al reflejar las tres palabras en el espacio se observa con claridad cómo "casa" y "apartamento" están mucho más cerca entre sí que cualquiera de ellas respecto a "playa", y, sin embargo, "apartamento" está más cerca de playa que "casa".

No obstante, la distancia entre las tres palabras no es mucha, si se representase "arsénico" u "oxidación" probablemente estarían muy alejadas de las tres palabras del ejemplo.

Los word Embeddings. Su origen

Bien, el ejemplo gráfico explica con claridad el concepto, aunque en la realidad los vectores tienen una dimensión mucho mayor, el concepto es el mismo. El vector de un token no solo describe a la palabra, sino que además palabras relacionadas se sitúan próximas en el espacio euclídeo n-dimensional en el que pueden representarse.

Más formalmente puede afirmarse que los Word Embeddings[1] son representaciones vectoriales de palabras, y representan un enfoque de la semántica de distribución que permite definir palabras a partir de un vector de números reales. Estos números representan características de las palabras, y por lo tanto los vectores permiten agrupar con similitudes semánticas o sintácticas. Tal y como explica la figura anterior.

Estas técnicas surgieron con fuerza en la década de los noventa, y desde entonces se han desarrollado diferentes formas y modelos para estimar representaciones continuas de palabras. Un ejemplo de esto es el análisis semántico latente o LSA.

La semántica distribucional

Dedicada a la cuantificación y categorización de las similitudes semánticas entre todo tipo de tokens o elementos de la lengua a partir de las propiedades distribucionales de los mismos con volúmenes de datos elevados. La hipótesis principal de la semántica distribucional nos conduce a afirmar que, términos con distribución espacial similar van a tener un significado parecido.

Por ejemplo, centrémonos en las expresiones "playa", "bañador" y "jersey". Bien, pues las dos primeras tendrán una distribución numérica muy similar, mientras que la tercera tendrá una distribución numérica diferente. Y, en efecto, cuando me refiero a distribución numérica estoy pensando en sus vectores descriptores, que pueden representarse en un mismo espacio n-dimensional.

[1]Término acuñado por Y. Bengio et al en 2003 en su artículo "*A Neural Probabilistic Languaje Model*", pero fue a partir de 2008, gracias a un texto de Collobert y Weston [7]. No obstante, hasta 2013, con la publicación de word2vec, no empezaron a utilizarse ampliamente.

Así pues, cuando en lugar de un vector tridimensional nos centramos en una representación densa de cada palabra (pongamos 150, 300 o más dimensiones) si que es posible conseguir que palabras con significados similares tengan distribuciones semejantes.

No es ámbito de la asignatura actual, pero no puedo dejar de mencionar que, a día de hoy, entendiendo los embeddings como vectores descriptores, no solo aplican a palabras, podemos encontrar embeddings de audio o de imágenes, … pero como decía al principio de este párrafo, es harina de otro costal, centrémonos en el procesamiento de texto.

Del mismo modo, no es objetivo de esta asignatura profundizar en embeddings, sino explicarlos al alumno por ser la base de modelos mucho más avanzados a día de hoy, por lo que no se va a entrar en detalles de uso de librerías de embeddings, **siendo esto, una tarea propuesta al alumno**.

En cualquier caso, se debe tener en cuenta que, lo que se consigue con los embeddings son dos cosas:

- codificarlas de modo que esta codificación represente el significado de cada una
- poder procesar esta codificación mediante aprendizaje automático para aprender la relación entre palabras.

Una decisión que se deberá tomar cuando se trabaje con embeddings es si entrenar un modelo nuevo o, por el contrario, emplear un modelo preentrenado. Cuando se trata de proyectos generalistas, la decisión es sencilla, pues existen modelos muy útiles, como algunos de los que vamos a ver. Por el contrario, si se tratase de un proyecto para un sector específico -medicina, derecho, etc...- quizá pueda resultar interesante entrenar un modelo, si bien, personalmente recomendaría, primero, probar con un modelo preentrenado.

Word2vec

Tal y como ya se ha indicado, aunque los inicios de los word embeddings datan del año 2003, cuando Bengio y sus colegas publicaron "*A Neural Probabilistic Language Model*", no es hasta 2013 año en que, un equipo de investigadores de Google publican word2vec, que los word embeddings no se empiezan a utilizar de forma amplia. Puedes consultar en https://arxiv.org/abs/1301.3781 el documento al que me refiero.

De forma muy resumida, puede afirmarse que word2vec se basa en dos técnicas:

- skip grams
- bag of words

El segundo término es una representación mediante codificación OneHot del vocabulario, mientras que, los skip-grams vamos a explicarlos de forma resumida a conticnuación, de modo que, el lector, comprenda las bases del modelo neuronal word2vec.

Skip – grams

Es una técnica para recorrer paso a paso un texto, en el que contamos con dos elementos, el puntero y la ventana:

- Puntero: Es el indicador que va saltando de palabra en palabra, apuntando en cada instante de tiempo a la palabra que se procesa

- Ventana: es el número de palabras (tamaño de ventana) que se tienen en cuenta antes y después de la palabra marcada por el puntero en cada time step, y que se van a considerar en el análisis y procesamiento de la palabra.

Por ejemplo, veamos un texto y cómo se procesaría a nivel skip gram con ventana de tamaño 2:

Texto: “Creo que mañana va a hacer frío”.
Puntero: Resaltado en amarillo.
Ventana: Resaltada en azul.

Time step	Texto	Skip gram – pares de entrenamiento
1	Creo que mañana va a hacer frío	(Creo, que), (Creo, mañana)
2	Creo que mañana va a hacer frío	(que, Creo), (que, mañana), (que, va)
3	Creo que mañana va a hacer frío	(mañana, Creo), (mañana, que), (mañana, va), (mañana, a)
4	Creo que mañana va a hacer frío	(va, que), (va, mañana), (va, a), (va, hacer)
5	Creo que mañana va a hacer frío	(a, mañana), (a, va), (a, hacer), (a, frío)
6	Creo que mañana va a hacer frío	(hacer, va), (hacer, a), (hacer, frío)
7	Creo que mañana va a hacer frío	(frío, a), (frío, hacer)

Ahora, para entrenar nuestro modelo neuronal word2vec, se emplearán as codificaciones bag of words (OneHot) de cada término de los pares.

Este modelo, en cuyo detalle no vamos a entrar, es capaz de generar para cada expresión multitud de relaciones con otras palabras, y representarlas en vectores de (habitualmente) 300 dimensiones reales.

A continuación, vamos a explorar diferentes librerías de embeddings, ya que hoy hay muchas, si bien vamos a empezar con un pequeño ejemplo de word2vec que es, cuando menos, curioso, pero que nos va a permitir comprender cómo funcionan los embeddings.

Cosine similarity

Cuando trabajamos con vectores, se puede comparar la similaridad entre ellos de diferentes formas, si bien la más extendida es la similitud del coseno, o cosine similarity, sobre la cual, aunque no vamos a entrar en detalles, ya que la implementan muchas librerías, se basa en la siguiente formulación:

Dados:

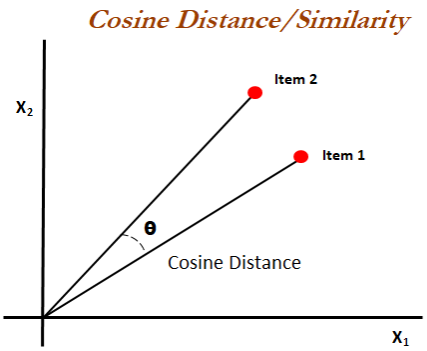
$A = vectordenúmeros, a_i = elementoideA$y
$B = vectordenúmeros, b_i = elementoideB$, sea

$$cosineSimilarity(A,B) = \frac{\sum_1^N a_i * b_i}{\sqrt{\sum_1^N {a_i}^2} * \sqrt{\sum_1^N {b_i}^2}}$$

Por cierto, no te preocupes, no vas a tener que implementar tú una función que realice el cálculo, son muchas las librerías que ya lo implementan. Por ejemplo, la librería scikit-learn trae una función

que, dada una lista de vectores, devuelve una matriz que compara cada vector con todos (la diagonal es 1 pues es la comparación de cada vector consigo mismo). Más adelante, en un ejemplo, la usaremos.

Gráficamente, puede representarse del siguiente modo:



Fuente: https://sitiobigdata.com/

Otra posibilidad sería, por ejemplo, emplear la distancia euclídea entre dos distribuciones para medir su semejanza, si bien *se suele emplear la similitud del coseno*. Una explicación de esto la puedes encontrar, por ejemplo, en este foro: https://www.quora.com/Why-do-we-use-cosine-similarity-on-Word2Vec-instead-of-Euclidean-distance

Existe la posibilidad, y encontrarás con facilidad muchas páginas web que explican cómo, de entrenar tu propio modelo word2vec, pero necesitarás una base de entrenamiento muy amplia (dataset válido). Un ejemplo muy bueno lo tienes en: https://keepcoding.io/blog/ejercicio-de-aplicacion-con-word2vec/. En resumidas cuentas, es una red neuronal de dos capas, nada especialmente complejo, y como ejercicio práctico te lo recomiendo si deseas entrenarte en el desarrollo de redes neuronales.

Ahora bien, para explorar las posibilidades de los embeddings, te recomiendo que te olvides de crear tu propio modelo word2vec y descargues alguno de los ya existentes, muy bien entrenados y listos para su uso. De la web de "***Spanish Billion Corpus***" puedes descargar un muy buen corpus de forma absolutamente gratuita. La web es: https://crscardellino.ar/SBWCE/. El nombre del modelo que te recomiento es "*SBW-vectors-300.min5.bin.gz*". Genera embedings de 300 dimensiones, algo más que suficiente en la mayoría de los casos, aunque si buscas en la web podrás encontrar modelos de 600 dimensiones, cuyo funcionamiento es, esencialmente, el mismo.

Existen numerosas librerías Python para generar embeddings. A continuación vamos a explorar algunos ejemplos.

Word2vec empleando gensim

Gensim es una librería desarrollada en lenguaje Python de "open source" para el procesamiento de lenguaje natural. Se define como "modelado temático para humanos". Fue desarrollada *Radim Rehurek*, conocido investigador PLN. No es una biblioteca de investigación de PNL que incluya un espectro amplio de funcionalidad, como puede ser el caso de NLTK, sino que más bien consiste

en un conjunto funcional práctico que ofrece bastantes posibilidades en el desarrollo de aplicativos que tienen como base el procesamiento del lenguaje natural.

En el caso que nos ocupa, debemos destacar que soporta una implementación de la incrustación de palabras de Word2Vec para el aprendizaje de nuevos vectores de palabras a partir de texto. También proporciona herramientas para cargar incrustaciones de texto pre-entrenadas en diferentes formatos y, posteriormente, poderlas emplear.

Veamos a continuación un ejemplo de uso de embeddings word2vec en español con la librería Gensim.

Notas:

- *Esta asignatura no pretende explicar en profundidad una librería -existe mucha documentación en internet acerca de cualquiera de ellas-, sino que se servirá de diferentes librerías para explicar cómo explotar algunas funcionalidades concretas.*
- *Este ejemplo, debido a que el modelo es muy pesado (1.1Gb) no lo probaremos en Collaboratory, sino en local, ya que puede tardar mucho en subirse el fichero con el modelo y generar errores.*
- *De hecho, cuando subas cualquier archivo de gran tamaño a un servidor en la nube (como por ejemplo collab) asegúrate de que se ha subido completamente antes de utilizarlo. En ocasiones la subida es lenta y, si vas a cargarlo en memoria es posible que intentes cargarlo antes de haberse subido por completo.*

Puedes encontrar el código fuente y el modelo en el siguiente repositorio de GitHub: https://github.com/jmolina010/Embeddings_Word2Vec_Gensim

Explicación del código:

El proyecto se compone de dos ficheros de ejemplo:

1. comparacion_de_vectores.py: Primer ejemplo, compara embeddings de palabras, mostrando las matrices de similitud
2. operaciones_con_vectores.py: Segundo ejemplo, ilustra cómo realizar operaciones entre palabras y luego las compara con otras, para demostrar hasta qué punto cómo los embeddings mantienen el significado cuando de forma lógica se realizan operaciones entre ellos.

El resto de los ficheros del proyecto son:

- requirements.txt: requerimientos del proyecto, las librerías necesarias que no se incorporan de forma estandard en el lenguaje Python
- SBW-vectors-300-min5.bin.gz: el corpus a emplear

comparacion_de_vectores.py

La función "*cargar_modelo*" únicamente carga en memoria el modelo word2vec a partir del corpus descargado y lo devuelve, de tal modo que se pueda emplear en el código.

La función "*obtener_vector*", dada una palabra y un modelo, devuelve el embedding de la misma siempre que ésta exista en el corpus.

Una vez cargado el modelo, hay dos bloques de código que generan los embeddings de una lista de palabras y luego emplean el método de la similaridad del coseno para obtener una matriz de similaridad. Puede observarse que es una matriz identidad, cuadrada y diagonal, ya que cada elemento es el coeficiente de similitud de un elemento con el resto de los mismos:

```
Comparacion de los embeddings de ['beagle', 'labrador', 'retriever', 'perro', 'dálmata', 'lobo', 'hiena'].
[[1.    0.478 0.738 0.663 0.419 0.432 0.403]
 [0.478 1.    0.557 0.587 0.316 0.411 0.324]
 [0.738 0.557 1.    0.693 0.406 0.437 0.421]
 [0.663 0.587 0.693 1.    0.419 0.622 0.531]
 [0.419 0.316 0.406 0.419 1.    0.333 0.314]
 [0.432 0.411 0.437 0.622 0.333 1.    0.611]
 [0.403 0.324 0.421 0.531 0.314 0.611 1.   ]]
```

Fast Text

Librería liberada por Meta (Facebook) hace ya tiempo (se desarrolló en el año 2016 y puede entenderse como una evolución de Word2Vec, además, está basada igualmente en Bag of Words y skipgrams). El paper original que describe el método empleado puedes localizarlo en https://arxiv.org/pdf/1607.04606.pdf, pero igualmente se incluye en la biblioteca básica de la asignatura.

En el repositorio https://github.com/dccuchile/spanish-word-embeddings se pueden encontrar corpus para FastText, Word2Vec (el ya conocido SBWC) y Glove (que se presenta más abajo).

	Corpus	Size	Algorithm	#vectors	vec-dim	Credits
1	Spanish Unannotated Corpora	2.6B	FastText	1,313,423	300	José Cañete
2	Spanish Billion Word Corpus	1.4B	FastText	855,380	300	Jorge Pérez
3	Spanish Billion Word Corpus	1.4B	Glove	855,380	300	Jorge Pérez
4	Spanish Billion Word Corpus	1.4B	Word2Vec	1,000,653	300	Cristian Cardellino
5	Spanish Wikipedia	???	FastText	985,667	300	FastText team

También pueden descargarse modelos desde https://fasttext.cc/docs/en/english-vectors.html, la página oficial de la librería fasttext.

De hecho, se pueden explotar modelos FastText desde la librería Gensim, así como desde la librería fasttext, entre otras (esta última proporciona modelos para más de 130 idiomas diferentes que pueden descargarse y descomprimirse automáticamente desde la propia librería), si bien, es posible que, dado que son modelos muy grandes y que contienen datos adicionales a los modelos Word2Vec, en un ordenador personal se aborte el proceso por problemas de memoria.

Debido a ello, se sugiere como forma de implementación en este caso, de la descarga del formato "*vec*", una versión utf-8 encoded que ya contiene el modelo compilado y que se puede cargar en memoria abriendo el fichero en forma tradicional y generando un diccionario de los embeddings (ver el código para más detalles)

El alumno puede encontrar el ejemplo en el repositorio siguiente: https://github.com/jmolina010/embeddings_fasttext_ejemplos, por lo que no va a profundizarse más en el código.

A excepción de la función de carga del modelo, el lector puede observar que el código es idéntico al anterior (word2vec con gensim). Excepto, obviamente, que no se utiliza la librería gensim.

Glove

GloVe (Global Vectors for Word Representation) es un sistema de aprendizaje no supervisado, que en 2014 desarrollaronpor Jeffrey Pennington, Richard Socher y Christopher D. Manning, siendo investigadores de la universidad de Stanford. Este sistema obtiene representaciones vectoriales de las palabras que recibe como entrada. Su entrenamiento se realiza sobre medidas de co-ocurrencia de palabras que se obtienen de un corpus, y las representaciones resultantes muestran estructuras lineales del espacio vectorial de las palabras de análisis.

La página oficial del algoritmo es https://nlp.stanford.edu/projects/glove/ y en ella puedes encontrar toda la información, últimos desarrollos y mucho material al respecto, si bien vamos a proceder a explicar de forma resumida los detalles fundamentales a continuación.

Para su ser entrenado, el modelo GloVe precisa de una matriz global de co-ocurrencia de palabras en la que cada celda indica cuán frecuentemente una palabra co-ocurre con otra (fila/columna) en un corpus determinado. Su objetivo de entrenamiento es aprender vectores de palabras tales que su producto punto sea igual al logaritmo de la probabilidad de co-ocurrencia de las palabras. El concepto subyacente es que la observación de ratios de co-ocurrencia de palabras tienen el potencial de codificar de alguna manera el significado de ellas.

Dicho de otro modo, y a diferencia de word2vec, que es un modelo predictivo, es decir, basado en técnicas de Machine Learning, y que implementa dos modelos neuronales (CBOW y Skip-gram, como ya se ha indicado) GloVe implementa un enfoque basado en conteo. GloVe genera una matriz de dimensiones “VxV” (donde V es la cantidad de palabras diferentes en el corpus) en la que
almacena la información de concurrencia entre palabras y contextos: para cada palabra se cuentan las apariciones en cada contexto. El objetivo de esta matriz no es otro que determinar vectores de palabras, basándose en la ocurrencia de las palabras en cada contexto. GloVe utiliza el aprendizaje
automático basándose en esta matriz. Por lo tanto, es un modelo híbrido (factorización matricial y predictivo).

GloVe se impolementó partiendo, como base, del artículo de Pennington et al. https://aclanthology.org/D14-1162.pdf, el cual también forma parte de la bibliografía esencial de la asignatura.

Cabe destacar, no obstante, que existen trabajos de investigación, como el que se puede encontrar en
https://addi.ehu.es/bitstream/handle/10810/29088/MemoriaTFG_IkerGarciaFerrero.pdf?sequence=3&isAllowed=y, que apuntan a que GloVe no escala bien a partir de corpus que
tengan un número determinado de palabras (por ejemplo, de 42x109 de tokens a 840 x109 de tokens
la mejora no es significativa). En este caso se probará con un corpus de GloVe, pero cargándolo desde word2vec, ya que ofrece esta posibilidad, como se podrá comprobar en el siguiente ejemplo.

Existen diferentes formas de utilizar los modelos vectoriales preentrenados de GloVe:

1. Empleando una librería que soporte el formato GloVe

2. Usando Gensim, pues se puede "exportar" el formato a word2vec y emplearlo como en el caso de word2vec
3. Leyendo los vectores y generando un diccionario, tal y como hemos hecho en el caso de FastText. Vamos a emplear este método.

- En primer lugar, deberán descargarse los vectores pre-entrenados. Para el ejemplo que nos ocupa es suficiente con el modelo GloVe de "dccuchile" -revisar el caso de FastText para acceder a la página de descarga-. Descargaremos el formato "vec".

Puedes localizar un ejemplo en: https://github.com/jmolina010/embeddings_glove_ejemplos

Otras técnicas de embeddings

Existen otras técnicas de embeddings, como por ejemplo TF-IDF, que el docente recomienda encarecidamente al alumnado estudiar, si bien no es el propósito de esta asignatura -NLP- dar cobertura a todas, sino mencionar algunas de las más populares, por lo que no se va a dedicar más espacio a explicar ni TF-IDF ni otras, si bien si que se recomienda que el alumno que profundice en otras técnicas de vectorización.

La utilidad de los embeddings va más allá…

Hasta ahora todo pareced muy interesante, pero … ¿imaginas la posibilidad de generar corpus específicos para un sector o una actividad? Por ejemplo, un corpus de vocabulario jurídico …

Si, es posible y en la red encontrarás muchos ejemplos.

Esto permitiría guardar los vectores preentrenados en el formato que prefieras y, haciendo uso de una librería, descubrir un mundo repleto de posibilidades. Y es que, hasta no hace mucho tiempo, los embeddings eran la herramienta más utilizada en NLP, por lo que era muy importante disponer, en algunas ocasiones, de modelos preentrenados para temas concretos, alcanzando de este modo una precisión mucho mayor que con un corpus genérico.

Ejercicios prácticos

Ejercicio resuelto. Creación de una red LSTM con la librería Keras

Tal y como prometimos, a continuación, vamos a presentar un ejemplo de red LSTM empleando embeddings, para que veas una aplicación nueva de los vectores.

Eso sí, antes, y llegados a este punto, debemos hacer memoria y repasar cómo, en el tema anterior, al implementar una red neuronal recurrente -RNN- tradicional, generábamos un diccionario en el que cada término del vocabulario tenía asociado un número (una especie de embedding unidimensional), por lo que, aunque implementando en este caso una red LSTM, no habrá grandes diferencias respecto al ejercicio resuelto del tema anterior, solo que, empleando las capas implementadas en Keras, podemos implementar una capa de tipo "*Embedding*" sin complicarnos más allá de instanciar la capa en nuestra arquitectura de red.

La librería Keras.

Se diseñó con el objetivo de simplificar el uso de la librería TensorFlow. Su creador, François Chollet[2], ha conseguido crear una librería de aprendizaje profundo de alto nivel que permite, incluso, redes de grafos y posibilita que el ingeniero de IA se pueda centrar mucho más en la arquitectura de la red que en el código de bajo nivel, simplifica,do, además, la comprensión estructural de los modelos.

Chollet, en su libro "*Deep Learning with Python*" dedica un capítulo -el número 11, titulado '*Deep Learning for text*') al procesamiento del lenguaje natural con aprendizaje profundo. Este libro es una auténtica joya que te recomiendo que adquieras y estudies en detalle. El autor tiene la capacidad de explicar de forma muy sencilla los conceptos matemáticos avanzados que envuelven a las redes neuronales, proporcionando una visión realmente comprensible del aprendizaje profundo.

Dado que el modelo a entrenar presenta una cierta complejidad, cuando ejecutes el ejemplo será muy recomendable que lo lances en Collaboratory activando GPU en el entorno de ejecución.

El código fuente, que viene comentado y explicado sobre la marcha, pero que, si el lector ha seguido las recomendaciones de estudio de Keras debería resultar muy sencillo, se puede descargar desde https://github.com/jmolina010/ejemplo_LSTM_Keras_Quijote/.

A continuación, se ponen capturas del código de la última celda y la ejecución de la misma.

```
# Ejemplo de generación

print('Texto generado sin demasiada creatividad:')
print('------------------------------------------------------------')
texto = generar_texto(modelo_ok, cadena_inicio='Rocinante', temperatura=0.3)
print(''.join(car for car in texto))
print()


print('Texto generado con una palabra que no está en el vocabulario:')
print('------------------------------------------------------------')
texto = generar_texto(modelo_ok, cadena_inicio='red neuronal', temperatura=0.3)
print(''.join(car for car in texto))
print()


print('Dotemos al generador de mayor creatividad...:')
print('------------------------------------------------------------')
texto = generar_texto(modelo_ok, cadena_inicio='escudero', temperatura=1)
print(''.join(car for car in texto))
print()
```

[2]https://fchollet.com/

```
Texto generado sin demasiada creatividad:
------------------------------------------------------------
, y como él se imaginaba que
había de hacer en el fragies y se lo dijo el mandamiento de su
deseo; que, puesto que los conocían que entre los dos se
dice que le he menester para condesa para el juez quién
ha sido la suya. En resolución, todos los de la carreta, donde le dejaremos por
la barca y voto a nosotros el vestido, por lo menos en más estraña catálantida. El cura los tambobernador mío, que no le quiero dejar de ser
tanto como el precio de la jineta un personaje, que todo lo demás pu

Texto generado con una palabra que no está en el vocabulario:
------------------------------------------------------------
 de los Amadís de Gaula, que sería destruido de cabellos
andantes, que se lo avisé, con que se lo agradece, y que, sin
ser alcanzare la verdad de lo que más le fatigaba era de comer a la ciudad, al entrar de
su palafrén, acomodado y contento, y, puesto en pie, y, después de haberla ella
hacerlas en cobrar en una redoma del cura, y desde allí a poco le aguardaba un bosque sus razones de don Quijote, el cual estaba
diciendo:

— Agora quiero decir, Sancho, que tiene por costumbre de ver que

Dotemos al generador de mayor creatividad...:
------------------------------------------------------------
 de los Espejos dieron a ruedores transfírmas y honestos agüeros que tornalmente de los
libros de caballerías, atómico y fuele de lleno infinito de serlo puñada y anotómete; porque si la cenarlo
mismo, que tengo góner con una venta, que le sucedió a don Quijote con una caña de maestresala, y, sobre
el otro, echados y revolviéndose a su cargo el escrucientos caballeros, que
brecientos iba de acuste y andamos, que no me harán
sartísimos principales comenzó a comer, no toparé a Dulcinea que no
```

Puede observarse que genera texto con bastante sentido (no siempre, ya que se trata de una red sencilla) teniendo en cuenta que el vocabulario es el del libro mencionado.

Puede apreciarse, igualmente, que cuando la cadena de entrada no aparece en el vocabulario, la red sigue generando texto, pero dentro del contexto aprendido.

Ejercicios propuestos

Cojo ejercicios se proponen cambios sobre el ejemplo anterior, de tal modo que el alumno pueda profundizar en el código de ejemplo:

- Modificar hiperparámetros de la arquitectura de red (función de optimización, función de pérdida, etc.)
- Modificar la longitud de las secuencias
- Entrenar la red con un libro de inteligencia artificial o con textos de wikipedia sobre computación cuántica

¿Qué cambios observas en la respuesta de la red neuronal al modificar los hiperparámetros y longitud de las secuencias?
¿Cómo afecta el contexto de entrenamiento al contenido generado?

Bibliografía para completar el aprendizaje

1. Vivky Boykis. *"Wath are Embeddings":* https://vickiboykis.com/what_are_embeddings/
2. García Ferrero, I. (2018). "*Estudio de Word Embeddings y métodos de generación de Meta Embeddings"* [Trabajo Final de Grado]: https://addi.ehu.es/bitstream/handle/10810/29088/MemoriaTFG_IkerGarciaFerrero.pdf?sequence=3&isAllowed=y
3. Pennington et al, J. (2014). "*GloVe: Global Vectors for Word Representation"* (Association for Computational Linguistics ACL, Ed.): https://www.aclweb.org/anthology/D14-1162.pdf.
4. Mikolov et al, (2013) "*Efficient Estimation of Word Representation in Vector Space*": https://arxiv.org/abs/1301.3781.
5. "*Word2vec y embeddings"*: https://sitiobigdata.com/2019/05/01/extraer-conocimiento-de-graficos-de-conocimiento/#

6. Piotr Bojanowski et al. (2017). "*Enriching Word Vectors with Subword Information*": https://arxiv.org/abs/1607.04606
7. "*Spanish Billion Corpus*": https://crscardellino.ar/SBWCE/.
8. Entrenamiento de un modelo word2vec propio: https://keepcoding.io/blog/ejercicio-de-aplicacion-con-word2vec/.
9. Paquete Gensim para PLN: https://radimrehurek.com/gensim/.