GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

TRABAJO FIN DE GRADO

Robot poeta. Selección y abstracción de un corpus literario en un modelo de deep learning

Autor: Carlos Geûens Álvarez

Director: Álvaro López López

Madrid

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título

Robot Poeta. Selección y abstracción de un corpus literario en un modelo de Deep Lear Ning, en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el

curso académico 2019/2020 es de mi autoría, original e inédito y

no ha sido presentado con anterioridad a otros efectos.

El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido

tomada de otros documentos está debidamente referenciada.

Fdo.: Carlos Geûens Álvarez Fecha: 18/ 07/ 2020

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

Fdo.: Álvaro López López Fecha: 19/07/2020

GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

TRABAJO FIN DE GRADO

Robot poeta. Selección y abstracción de un corpus literario en un modelo de deep learning

Autor: Carlos Geûens Álvarez

Director: Álvaro López López

Madrid

ROBOT POETA. SELECCIÓN Y ABSTRACCIÓN DE UN CORPUS LITERARIO EN UN MODELO DE DEEP LEARNING

Autor: Geûens Alvarez, Carlos.

Director: López López, Alvaro.

Entidad Colaboradora: ICAI – Universidad Pontificia Comillas

RESUMEN DEL PROYECTO

El proyecto consiste en la realización de una inteligencia artificial entrenada con un corpus literario del poeta Vicente Aleixandre, para que sea capaz de realizar su propios versos y estrofas. Este proyecto se centrará en el tratamiento del corpus literario, su implementación en una red neuronal LSTM, la obtención de unos primeros versos coherentes y su implementación en un bot de Twitter.

La problemática planteada consiste en lograr tratar computacionalmente poesía una expresión artística y abstracta completamente ligada a la creatividad y a la inteligencia lingüística-verbal del ser humano.

Para la creación de inteligencias artificiales que hagan frente a este tipo de problemas se hace uso de las redes neuronales Son redes formadas por un conjunto de neuronas artificiales con entradas procesadas a través de un algoritmo para la obtención de una salida comunicándose con el resto de las neuronas. Haciendo posible el tratamiento de problemas abstractos en los que no existe un claro algoritmo para tratarlos mediante otros métodos computacionales, es el caso del lenguaje.

Existen numerosos tipos de red neuronales según el problema que se quiera resolver. Para saber que red neuronal es la más apropiada para un proyecto ha de analizarse el problema en cuestión y ver qué tipo de lógica será más adecuada para su abstracción. En el caso particular del tratamiento de un corpus literario, la principal característica consta en su carácter recurrente. Podemos pensar en cada palabra como una salida que depende de la salida/palabra anterior. De esta manera, de una forma ideal, esperaríamos que, si la palabra anterior es “los”, la subsiguiente palabra que depende de esta palabra además del entrenamiento y parámetros de la red neuronal sea “perros” y no “perro”. Con esta lógica la red recurrente es capaz de crear recurrentemente palabra a palabra una frase completa. Sin embargo, lenguaje es un concepto abstracto que no puede simplificarse tan fácilmente, y de la misma forma en que las palabras posteriores en una frase dependen de las anteriores, las anteriores pueden depender de las posteriores. La comunicación de ideas a través del lenguaje es complicada y esto también ha de ser analizado y tenido en cuenta a la hora de crea la red neuronal.

No solo es importante elegir el tipo de red neuronal pero también elegir correctamente el conjunto de entrenamiento y tratarlo correctamente. No podemos coger directamente y de cualquier manera la obra y entrenar a la red neuronal. Debemos traducirla a un lenguaje que entienda y pueda tratar la inteligencia artificial. En el caso de tratamiento del lenguaje, se debe tratar cada palabra como una entrada que pueda introducirse en la red neuronal y que esta sepa exactamente de qué palabra se trata.

Pero no solo hay que por convertir el conjunto de entrenamiento en algo que la red neuronal pueda comprender. También se debe analizar el tratamiento del conjunto de entrenamiento. Es decir, que versos seleccionaremos y como los dividiremos y trataremos para que al introducirlos en nuestra red neuronal aprenda correctamente y una vez realizado el proceso de entrenamiento seamos capaces de crear versos creativos y únicos.

Para realizar el proyecto, Se seleccionó el corpus literario Vicente Aleixandre obtenidos en https://www.poesi.as/Vicente_Aleixandre.htm. Una vez seleccionada la obra como dataset del proyecto se realizó el proceso de tokenización y tratamiento de la obra para convertirlo en algo que pueda ser comprendido por una red neuronal.

El primer paso fue la creación del diccionario. Para ello se fueron leyendo una a una de las palabras de los versos seleccionados creando una nueva entrada cada vez que se encontraba una palabra distinta, obteniendo así una lista de todas las palabras utilizadas por Vicente Aleixandre en su obra.

Una vez creado el diccionario se convirtió la obra en secuencias, formadas por un primer número indicando el inicio de verso, a continuación, el numero correspondiente a la posición en el diccionario de cada palabra del verso y finalmente un numero indicando el final de verso. Más tarde, también se implementó el inicio de estrofa y final de estrofa, para poder entrenar el modelo con estrofas completas en lugar de versos y por lo tanto también poder crear estrofas.

Para realizar correctamente el entrenamiento debemos tener la misma longitud de secuencias para que el input sea siempre de la misma dimensión y podamos introducirlo por lo que se alargaron las secuencias añadiendo el final de verso/estrofa hasta tener el tamaño de la secuencia más larga. Por último, para el entrenamiento del modelo es necesario transformar estas secuencias en binario (formato one-hot vector) para entrenar el modelo, de forma que cada palabra en lugar de ser un número es un 1 en la posición de un vector de tamaño el diccionario con el resto 0. Este proceso de tokenización se muestra en la Ilustración 1.

	Carlos	es	muy	guapo
posición palabra				
Carlos	1	0	0	0
es	0	1	0	0
muy	0	0	1	0
guapo	0	0	0	1
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.
numero_palabras	0	0	0	0

Ilustración 1:one-hot vector

Una vez transformado el dataset, se creó la red neuronal en Keras a entrenar con dicho Dataset. Se creo una red neuronal LSTM, red neuronal recurrente la topología mostrada en Ilustración 2. Esta estructura se explicará con más detalle en el apartado Redes neuronales y LSTM.

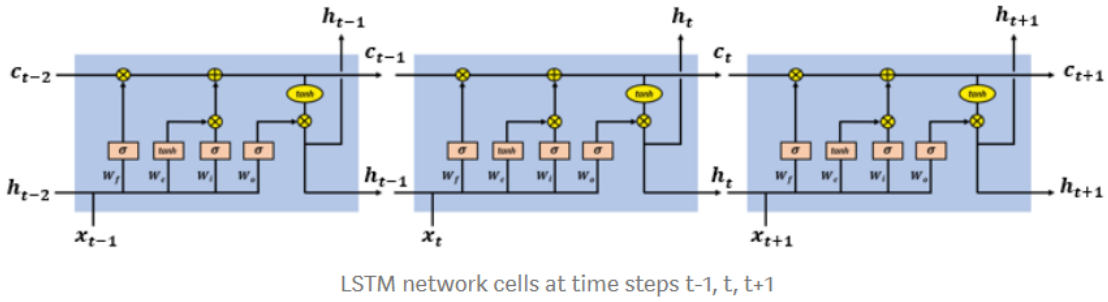


Ilustración 2: topología LSTM [1]

Una vez creada la red neuronal se entrenará con nuestro dataset, indicaremos a la red neuronal que queremos obtener como salida los vectores one-hot del dataset. La red neuronal modificara los pesos de la red LSTM para reducir el error, definido por la función de pérdidas de la ecuación Ecuación 8: función de perdidas categorical crossentropy y explicado con más de talle en Entrenamiento modelo.

Una vez entrenado el modelo, para generar texto se introduce al modelo como primera palabra la etiqueta correspondiente a inicio y creamos las secuencias a partir de ahí. Obtendremos al dar el modelo esta primera palabra, las probabilidades de todas las palabras que elegiremos de forma estocástica en lugar de determinista, no eligiendo siempre la palabra con mayor probabilidad. Este carácter estocástico se realiza con una temperatura en una función de sampling que nos indicará como de determinista será la secuencia que nos entregue el modelo. A partir de esta función de sampling se escogerá la siguiente palabra de la frase, que guardaremos con el resto de la frase que teníamos previamente para introducir nuevamente al modelo la frase con la nueva palabra y volver a samplear. Este proceso se realiza hasta que lleguemos a la etiqueta que nos indique el final de verso o estrofa y que significa que hemos llegado al final de la secuencia.

Cuando se halla obtenido un modelo que cree versos con cierta estructura gramatical y parecidos a poesía humana, se pasará a la creación del Bot de Twitter. Para ello se hará uso de la clave obtenida a través de Twitter developper, que nos permitirá acceder a la API de Twitter para desde Python y con la ayuda de la librería tweepy.org ir tuiteando los versos que vallamos obteniendo del modelo.

Tras realizar el proceso de tokenización y la red neuronal se obtuvieron unos versos decentes que podemos considerar poesía, confirmando la suposición inicial de que las redes neuronales actuales pueden crear texto lírico.

Un ejemplo de los versos creados son los de Ilustración 3.

```
rosa , robusta gastada o presentimiento ,
casi sagrada
sintiendo bóveda como La fría ,
como mirándome oigo .
```

Ilustración 3: versos modelo

También se demostró que este procedimiento podía aplicarse también a estrofas como la de Ilustración 4.

```
Manos ilustre humana como el amor ,
que nunca se ve pasar ,
el vacío verde o verde o lágrima
que a veces aquí se yergue a un beso en un agua .
```

Ilustración 4: estrofa modelo

La implementación del modelo al bot de Twitter también dio buenos resultados como se muestra en la Ilustración 5.

Ilustración 5. Bot Twitter

Por todo esto podemos afirmar que puede realizarse algo parecido a poesía con una red neuronal LSTM. Con el dataset y el entrenamiento necesario, se puede obtener algo que que parece entender la gramática y semántica española. Sigue habiendo errores de coordinación y la red neuronal no parece entender del todo algunos conceptos sintácticos. En cuanto al

nivel poesía también podría mejorar, aunque a veces nos encontramos con imágenes y frases interesantes, generalmente las frases no tienen el nivel de la poesía de Vicente Alixandre y nos encontramos con palabras repetidas y estrofas sin un tema demasiado coherente.

Para solucionar eso la primera opción es seguir parametrizando la red, buscar que profundidad es la más conveniente, si es mejor tener una red biridireccional etc, trabajo en el que se centró Venancio García en su proyecto paralelo. Para seguir impulsándolo también se podría aumentar el dataset, incluyendo a otros autores para tener más entrenamiento y que la red aprenda esos conceptos que sigue sin dominar. También se podría llevar a otro nivel la red neuronal, implementando una GAN, con un discriminador que intente distinguir entre versos reales y versos creados por nuestro modelo. Ajustando nuestro modelo para que cada vez engañe mejor al discriminador, y mejorando el discriminador para que también sea cada vez más eficaz. Podríamos de esta forma en teoría mejorar nuestro modelo mediante este enfoque competitivo, como ya se ha demostrado en inteligencias artificiales centradas en imágenes.

Referencias

[1] Nir Arbel, medium.com, Julio 2020, https://medium.com/datadriveninvestor/how-do-lstm-networks-solve-the-problem-of-vanishing-gradients-a6784971a577

ROBOT POET. SELECTION AND ABSTRACTION OF A LITERARY CORPUS IN A DEEP LEARNING MODEL

Author: Geûens Alvarez, Carlos.

Supervisor: López López, Alvaro.

Collaborating Entity: ICAI – Universidad Pontificia Comillas

ABSTRACT

The project consists of programming an artificial intelligence trained with poet Vicente Aleizandre ´s literary corpus, to make the inteligence able of creating its own verses. This project will be focused on the treatment of the literary corpus, its implementation in an LSTM neural network, obtaining some coherent first verses and its implementation in a Twitter bot.

The problem posed consists of computationally treat poetry, an artistic and abstract expression completely linked to the creativity and the linguistic intelligence of human beings.

To create artificial intelligences that face this type of problem, usually neural networks is the way to go. They are networks formed by a set of artificial neurons with inputs processed through an algorithm to obtain an output, communicating with the rest of the neurons. Making possible the treatment of abstract problems in which there is no clear algorithm, making it no posible to deal with them by other computational methods, its the case of language.

There are numerous types of neural networks depending on the problem we want to solve. To know which neural network is the most appropriate for a project, the problem to solve must be analyzed and we need to search which logic will be most suitable for its abstraction. In the case of the treatment of a literary corpus, the main characteristi consists in its recurring character. We can think of each word as an output that depends on the previous output / word. This way, ideally, we would expect that, if the previous word is “los”, the subsequent word that depends on this word in addition to the training and parameters of the neural network is “perros” and not “perro”. With this logic, the recurring network is able to recurrently create a complete sentence word by word. However, language is an abstract concept that cannot be so easily simplified, and in the same way that the later words in a sentence depend on the previous ones, the previous ones can depend on the later ones. The communication of ideas through language is complicated and this must also be analyzed and taken into account when creating the neural network.

Furthermore, it is not only important to choose the type of neural network but also to correctly choose the training set and treat it correctly. We cannot directly and in any way take the corpus and train the neural network. We must translate it into a language that the neural network will understand. In the case of language treatment, each word must be treated as an input that can be entered into the neural network so it knows exactly what word it is.

But to convert the training set into something that the neural network can understand is not the only thing that has to be taken into acccount. The treatment of the training set should also be discussed. That is, which verses we will select and how we will divide and treat them so that when we introduce them into our neural network, we learn correctly and once the training process has been completed, we will be able to create creative and unique verses.

To carry out the project, Vicente Aleixandre´s literary corpus obtained at https://www.poesi.as/Vicente_Aleixandre.htm was selected. Once the corpus was selected as the project's dataset, the process of tokenization and treatment of the work was carried out to turn it into something that can be understood by a neural network.

The first step was creating the dictionary. To do this, we read one by one all the words of the selected verses, creating a new entry each time a different word was found, thus obtaining a list of all the words used by Vicente Aleixandre in the dataset.

Once the dictionary was created, the corpus was converted into sequences, formed by a first number indicating the beginning of the verse, then the number corresponding to the position in the dictionary for each word in the verse and finally a number indicating the end of the verse. Later, the star of the stanza and the ending of the stanza were also implemented, to be able to train the model with complete stanzas instead of verses and therefore also be able to create them.

To correctly carry out the training we must have the same length in all sequences so that the input is always of the same dimensión. For this reason, the sequences were lengthened adding the end of verse / stanza until the size of the longest sequence. Finally, for the training of the model it is necessary to transform these sequences into binary (one-hot vector format) to train the model, so that each word instead of being a number is a 1 in the position of a vector of size dictionary with the remainder 0. This tokenization process is shown in Ilustración 6.

	Carlos	es	muy	guapo
posición palabra				
Carlos	1	0	0	0
es	0	1	0	0
muy	0	0	1	0
guapo	0	0	0	1
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.
numero_palabras	0	0	0	0

Ilustración 6: one-hot vector

Once the dataset was transformed, we created the neural network in Keras to be trained with said Dataset. An LSTM neural network was created, a recurrent neural network with the topology shown in Ilustración 7. This structure will be explained in more detail in the section Redes neuronales y LSTM.

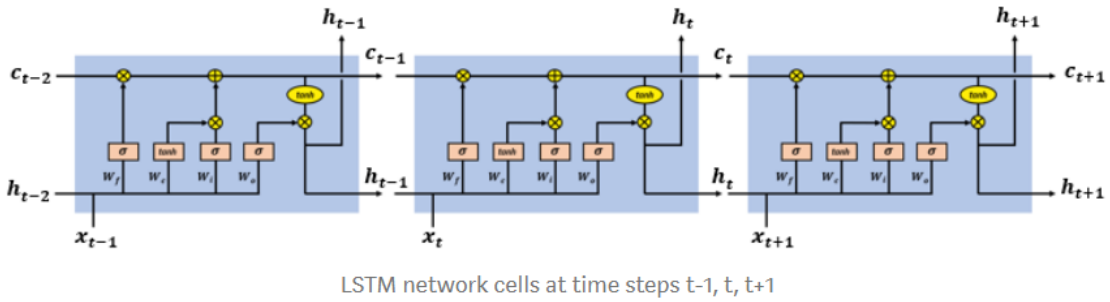


Ilustración 7: LSTM structure[2]

Once the neural network is created, it will be trained with our dataset, we will indicate to the neural network that we want to obtain the one-hot vectors of the dataset as output. The neural network will modify the weights of the LSTM network to reduce the error, defined by the los function of the equation Ecuación 8: función de perdidas categorical crossentropy and explained in more detail in Entrenamiento modelo.

Once the model has been trained, to generate text, the label corresponding to the beginning is entered as the first word and we create the sequences from there. By giving the model this first word, we will obtain the probabilities of all the words that we will choose stochastically instead of deterministically, not always choosing the word with the greatest probability. This stochastic character is performed with a temperature in a sampling function that will indicate how deterministic the sequence that the model gives us will be. From this sampling function, the next word of the phrase will be chosen, which we will save with the rest of the phrase that we previously had, to introduce the phrase with the new word to the model and sample again. This process is carried out until we reach the label that indicates the end of the verse or stanza and that means that we have reached the end of the sequence.

If we achieve a model that creates verses with a certain grammatical structure and similarity to human poetry, the creation of the Twitter Bot will be carried out. For this, the key obtained through Twitter developper will be used, which will allow us to access the Twitter API from within Python and with the help of the tweepy.org library, tweeting the verses obtained from the model.

After performing the tokenization process and the neural network, decent verses were obtained that we could consider poetry, confirming the initial assumption that current neural networks can create lyrical text.

An example of the verses created are those of Ilustración 8.

```
rosa , robusta gastada o presentimiento ,
casi sagrada
sintiendo bóveda como La fría ,
como mirándome oigo .
```

Ilustración 8:model verse

It was also shown that this procedure could also be applied to stanzas such as Ilustración 9.

```
Manos ilustre humana como el amor ,
que nunca se ve pasar ,
el vacío verde o verde o lágrima
que a veces aquí se yergue a un beso en un agua .
```

Ilustración 9: model stanza

The implementation of the model to the Twitter bot also gave good results as shown in the Ilustración 10.

Ilustración 10: Twitter Bot

For all this reasons, we can affirm that something similar to poetry can be done with an LSTM neural network. With the dataset and the necessary training, we can get something that seems to understand Spanish grammar

and semantics. Erros remain and the neural network does not seem to fully understand some syntactic concepts. Regarding the poetry level it could also improve, although sometimes we find interesting images and phrases, generally the phrases do not have the level of Vicente Alixandre's poetry and we find ourselves with repeated words and stanzas without a coherent theme.

To solve this, the first option is to continue parameterizing the network, looking for the depth that is the most convenient, if it is better to have a bi-directional network, etc., a work that Venancio García focused on in his parallel project. To continue improving, the dataset could also be increased, including other authors to have more training, so the network can learn those concepts that it still does not dominate. The neural network could also be taken to another level, implementing a GAN, with a discriminator that tries to distinguish between real verses and verses created by our model. Adjusting our model so that it increasingly deceives the discriminator, and improving the discriminator so that it is also increasingly more effective. We could in theory improve our model using this competitive approach, as has already been demonstrated in image-centric artificial intelligence.

References

[2] Nir Arbel, medium.com, Julio 2020, https://medium.com/datadriveninvestor/how-do-lstm-networks-solve-the-problem-of-vanishing-gradients-a6784971a577

Índice de la memoria

Índice de figuras

Índice de ecuaciones

Capítulo 1. INTRODUCCIÓN

El referente de la inteligencia artificial, Andrew Ngo, dijo en una de sus conferencias en Stanford MSx Future Forum [3], que la inteligencia artificial es la nueva electricidad. La electricidad cambió radicalmente todo el mundo en el que nos movemos, cambiando como entendíamos el transporte, la agricultura, la mecanización del campo, la industria y el mundo sanitario. De esta misma forma la inteligencia artificial puede ser aplicada a multitud de campos y aplicaciones pudiendo cambiar nuestra percepción del mundo en el futuro, con transportes autónomos, mecanizando la agricultura y la industria de una forma todavía más eficaz e independiente del ser humano y facilitándonos la detección y cura de enfermedades, como puede ser la detección precoz del cáncer de mama [4].

La inteligencia artificial tiene un valor añadido y es que no solo puede aplicarse a campos ya revolucionados por la electricidad y otras tecnologías, sino que puede aplicarse también a otros lugares cuyas características han dificultado la intromisión de las nuevas técnicas. Se trata de campos abstractos, creativos y que ligamos a la capacidad y sensibilidad humana, alcanzando su máxima expresión en el arte.

Existen ya herramientas para abstraer texto computacionalmente. En este proyecto se hará uso de estas herramientas aplicándolas a poesía en castellano con la intención de realizar una inteligencia artificial capaz de realizar sus propios versos y estrofas.

La intención es profundizar en este tipo de inteligencias artificiales, explorar sus capacidades y en definitiva si es capaz de realizar algo tan humano y creativo como es la poesía. Explorando al mismo tiempo cual es el valor de la poesía y suscitando preguntas sobre que es en realidad poesía y si realmente puede ser realizada por una entidad no humana.

Esta idea ha de ser llevada a la práctica y para ello debe ser estudiado el problema ante el que nos encontramos y las herramientas a nuestra disposición. La problemática planteada consiste en lograr tratar computacionalmente poesía una expresión artística y

abstracta completamente ligada a la creatividad y a la inteligencia lingüística-verbal del ser humano.

Para hace frente a este tipo de problemáticas se hace uso de inteligencias artificiales a través de redes neuronales. Redes formadas por un conjunto de neuronas artificiales con entradas procesadas a través de un algoritmo para la obtención de una salida comunicándose con el resto de las neuronas. Haciendo posible el tratamiento de problemas abstractos en los que no existe un claro algoritmo para tratarlos mediante otros métodos computacionales, es el caso del lenguaje.

Existen numerosos tipos de red neuronales según el problema que se quiera resolver. Para saber que red neuronal es la más apropiada para un proyecto ha de analizarse el problema en cuestión y ver qué tipo de lógica será más adecuada para su abstracción. En el caso particular del tratamiento de un corpus literario, la principal característica consta en su carácter recurrente. Podemos pensar en cada palabra como una salida que depende de la salida/palabra anterior. De esta manera, de una forma ideal, esperaríamos que, si la palabra anterior es “los”, la subsiguiente palabra que depende de esta palabra además del entrenamiento y parámetros de la red neuronal sea “perros” y no “perro”. Con esta lógica la red recurrente es capaz de crear recurrentemente palabra a palabra una frase completa. Sin embargo, lenguaje es un concepto abstracto que no puede simplificarse tan fácilmente, y de la misma forma en que las palabras posteriores en una frase dependen de las anteriores, las anteriores pueden depender de las posteriores. La comunicación de ideas a través del lenguaje es complicada y esto también ha de ser analizado y tenido en cuenta a la hora de crea la red neuronal.

No solo es importante elegir el tipo de red neuronal pero también elegir correctamente el conjunto de entrenamiento y tratarlo correctamente. No podemos coger directamente y de cualquier manera la obra y entrenar a la red neuronal. Debemos traducirla a un lenguaje que entienda y pueda tratar la inteligencia artificial. En el caso de tratamiento del lenguaje.

Pero no solo hay que por convertir el conjunto de entrenamiento en algo que la red neuronal pueda comprender. También se debe analizar el tratamiento del conjunto de entrenamiento. Es decir, que versos seleccionaremos y como los dividiremos y trataremos para que al introducirlos en nuestra red neuronal aprenda correctamente y una vez realizado el proceso de entrenamiento, este sea óptimo y seamos capaces de crear versos creativos y únicos.

Todo, este proceso ha de realizarse analizando y estudiando los resultados que nos valla proporcionando la red neuronal, para optimizar el procesado de datos y la parametrización de la red.

Una vez obtengamos nuestra inteligencia artificial, de nosotros depende el uso que le queramos dar, ya sea escribir una obra o para el estudio del autor de entrenamiento, que también se exploraran en este proyecto.

Capítulo 2. DESCRIPCIÓN DE LAS TECNOLOGÍAS

Se presentan las tecnologías utilizadas para la creación de la red neuronal su entrenamiento y su posterior uso para creación de obras e implementación en un bot de twitter.

2.1 REDES NEURONALES

Para la creación de la IA se creará una red neuronal LSTM a la que se entrenará con el corpus escogido. Para ello debemos entender primero que es una red neuronal.

Una **red neuronal** está compuesta por un conjunto de neuronas interconectadas entre sí mediante enlaces [5].

Cada neurona toma como entradas las salidas de las neuronas anteriores, cada una de esas entradas se multiplica por un peso, que se ajusta durante el proceso de entrenamiento. Todos los resultados parciales de las entradas multiplicadas por el peso se agregan y mediante una función de activación se calcula la salida. Esta salida es a su vez es entrada de la neurona a la que precede.

La función de activación se encarga de mantener el conjunto de valores de salida en un rango determinado. Existen distintas funciones de activación con características diferentes. La función de activación utilizada en el proyecto es la función softmax, sigmoid y tanh.

Softmax es una función que convierte un vector de números reales en un vector de números entre 0 y 1, que representa las probabilidades de cada número, y cuya suma total es 1. La ecuación de dicha función es Ecuación 1 [6].

Ecuación 1: Softmax

$$\sigma(\vec{z})_i = \frac{e^z i}{\sum_{j=1}^{k} e^{z_j}}$$

Donde Zi representa el valor de cada elemento del vector Z, obteniendo un valor exponencial para cada elemento, que puede ser reducida con un factor de temperatura dividiendo Zi en

la exponencial, y que dividimos por el sumatorio de todos estos exponenciales para tener un valor de probabilidad entre 0 y 1.

Sigmoid es una función de activación que limita los outputs entre 0 y 1. La ecuación de la sigmoidal es Ecuación 2:

Ecuación 2: Sigmoid

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

Finalmente, la **tanh**, de forma muy parecido a la sigmoid, pero comprendida entre -1 y 1. Cuya ecuación es Ecuación 3

Ecuación 3:tanh(x)

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

Siendo por la tanto la representación de la función sigmoid (en azul) y tanh (en naranja) las mostradas en la siguiente figura Ilustración 11.

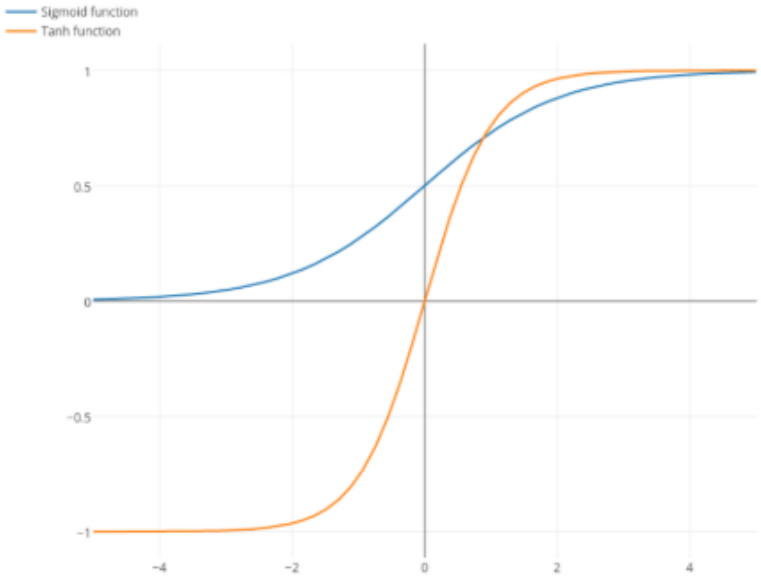


Ilustración 11: representación sigmoid y tanh. [7]

Al aplicar estas funciones de activación, obtenemos una estructura neuronal con unas neuronas de entrada, un conjunto de neuronas que formarán las capas ocultas y finalmente la capa de salida calculada por el conjunto de neuronas. Como puede verse en Ilustración 12: Representación estructura red neurona. Donde x representan las entradas, w los pesos de las salidas de cada neurona, a las neuronas intermedias e y las salidas finales de la red neuronal.

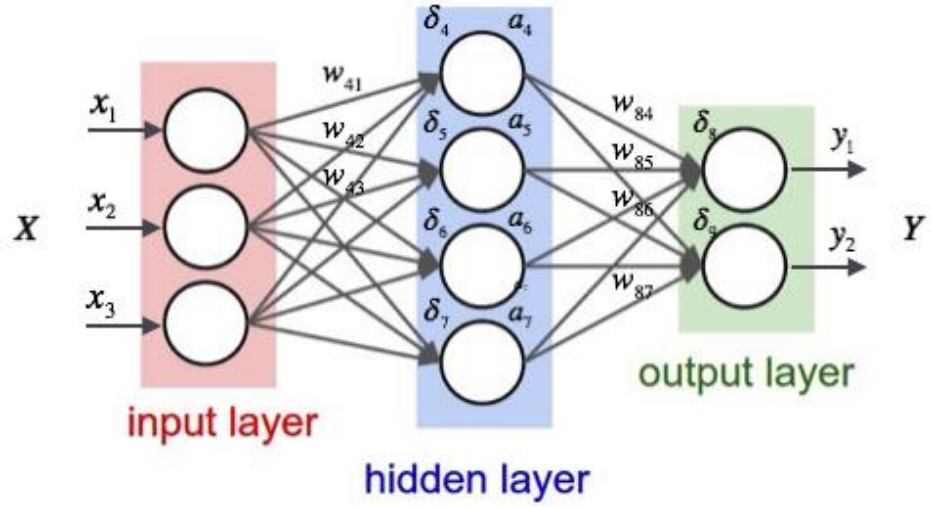


Ilustración 12: Representación estructura red neuronal [9]

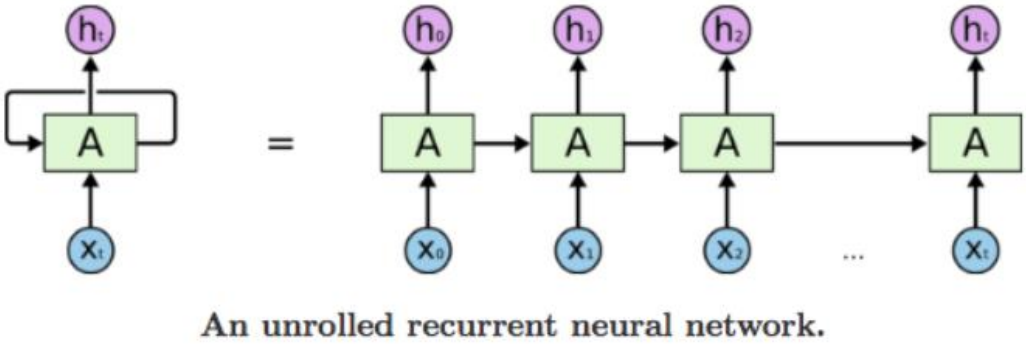
Una vez entendida esta estructura, solo falta mencionar de donde surgen el valor de los pesos que serán los causantes de que la salida sea aquello que esperamos. El valor de estos pesos se calcula en el proceso de entrenamiento. Este proceso es un proceso por repetición que realizaremos tantas veces como sea necesario. En este proceso de entrenamiento, tendremos acceso al dataset de entrenamiento lo que nos permite conocer el input y el output que debemos obtener dado dicho input. De esta forma durante el entrenamiento la red neuronal nos dará los outputs estimados que podremos comparar con los reales. Este error entre el valor estimado por la red y el valor real que debería de haber salido se calcula a través de una función de coste cuya intención es minimizar.

La red neuronal ira por lo tanto en cada repetición ajustando los pesos para que el error de la función de coste sea menor. La variación de estos y este proceso de ajuste se explica el apartado Entrenamiento modelo.

2.2 REDES NEURONALES Y LSTM

Sin embargo, no existe únicamente una estructura de red neuronal, y a partir de este concepto existen variantes que nos permiten afrontar de forma más eficaz el carácter recurrente de la poesía. Se trata de las **RNN** (recurrent neural networks).

Las RNN son una generalización de las redes neuronales que tienen la característica de tener una memoria interna, de forma que, una vez producido un output, este es copiado y reenviado a la red neuronal recurrente, permitiéndonos tener en cuenta el siguiente input y el output interior para realizar la próxima decisión [10]. Esto permite procesar secuencias de inputs con relaciones entre ellos creando ese carácter recurrente tan importante para nuestro proyecto y que se muestra esquematizado en Ilustración 13: Esquema RNN.



An unrolled recurrent neural network.

Ilustración 13: Esquema RNN. [10]

El inconveniente de las RNN es el problema de la desaparición del gradiente. Debido al amplio número de capas podemos encontrarnos con el problema de que, al aplicar reiterativamente la función de activación, el efecto de los pesos de las primeras capas desaparece exponencialmente en el proceso de entrenamiento, lo que dificulta mucho el entrenamiento de los pesos de las primeras capas.[11]

Una de las formas de resolver el problema de la desaparición del gradiente puede ser usar funciones de activación que no favorezcan tanto este proceso como la función Relu. Pero existe una red neuronal recurrente que intenta hacer frente a este inconveniente, y será la que utilizaremos en este proyecto, la red neuronal **LSTM**.

La red LSTM resuelve el problema de la desaparición del gradiente a través de tres puertas, input, output y forget gate. Esta última informa a las celdas de la LSTM que información

deben olvidar dado el nuevo input. De esta forma gracias a vector de estado c(t) que depende de esta froget gate, la red neuronal es capaz de controlar de forma más eficaz la información que debe recordar y saber que parámetros debe actualizar, reduciendo en gran medida el problema de desaparición del gradiente. [12]

La estructura de la LSTM es por lo tanto la mostrada a en Ilustración 14.

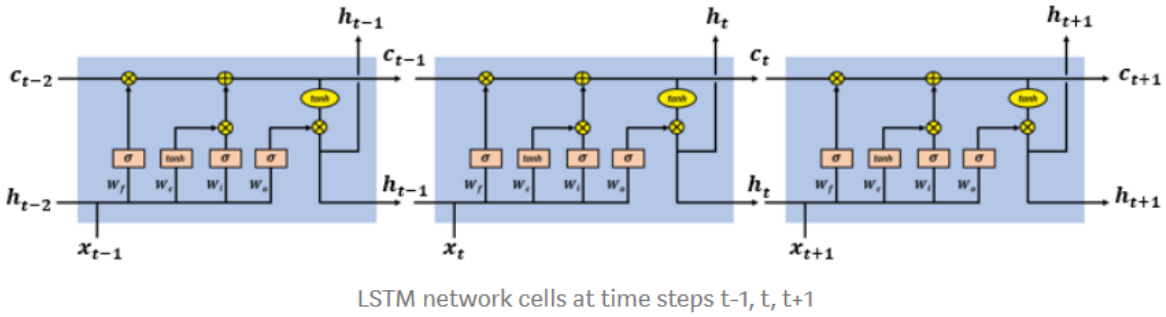


Ilustración 14: representación LSTM para iteración t-1, t, y t+1. [12]

De forma que c_t es el estado de la celda, h_t el output, x_t el input junto con h_{t-1}, el output de la celda anterior. Las W serán los pesos para cada una de las funciones internas, que serán tratados bien con la función de activación sigmoid explicada en Redes neuronales y cuya formula es Ecuación 2 o con la función de activación tanh con fórmula Ecuación 3.

De esta forma tendríamos un el output del **forget gate**, representando la información que debe olvidarse, con la siguiente Ecuación 4:

Ecuación 4: forget gate

$$f_t = \sigma\left(w_f \cdot [h_{t-1}, x_t]\right)$$

Que en la celda representaría el siguiente camino mostrado en Ilustración 15:

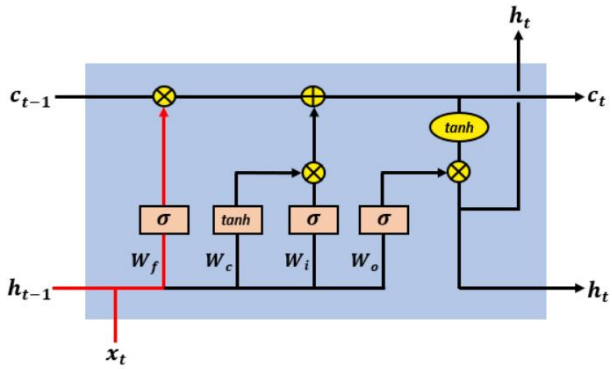


Ilustración 15: figura output gate LSTM. [12]

A continuación, tendríamos la **input gate** que controla que información debe ser recordada por en el estado de la celda, con ecuación Ecuación 5:

Ecuación 5: input gate

$$(\tilde{c}_t \otimes i_t) = \tanh(w_c \cdot [h_{t-1}, x_t]) \otimes (w_i . [h_{t-1}, x_t])$$

Camino representado en la Ilustración 16:

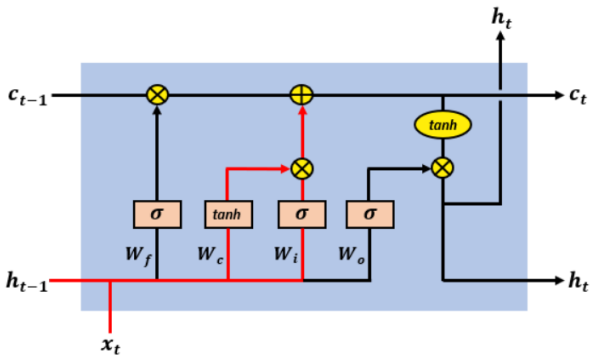


Ilustración 16: figura input gate LSTM. [12]

Finalmente tendríamos la **output gate**, de Ecuación 6:

Ecuación 6: output gate

$$h_t = \sigma(w_0 \cdot [h_{t-1}, x_\tau]) \otimes \tanh(C_t)$$

Y que representa el camino de Ilustración 17:

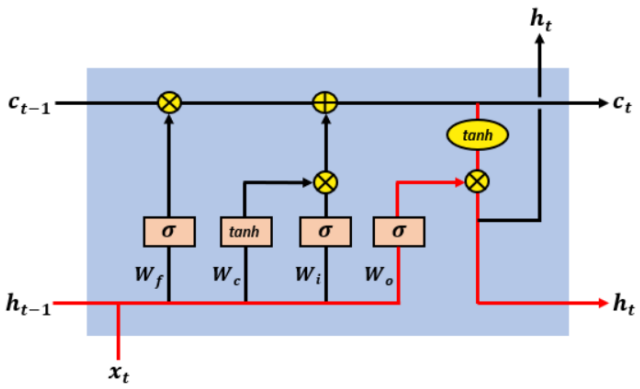


Ilustración 17: figura output gate LSTM. [12]

Por lo que tendríamos dos salidas de la celda h_t de Ecuación 6 y el estado que pasamos a la siguiente iteración c_{t-1} de Ecuación 7:

Ecuación 7: estado LSTM

$$c_t = c_{t-1} \otimes f_t \otimes (\tilde{c}_t \otimes i_t)$$

2.3 *TOKENIZACIÓN*

Tokenización es el proceso de convertir cualquier cosa en un elemento digital [13]. En el caso de nuestro proyecto se trata de convertir texto en input para introducirlos a la red neuronal.

Para hacer frente a este problema usaremos la codificación **one-hot,** en la que cada palabra se representa de forma binaria, con vector de 0 de tamaño el número de palabras, excepto un 1 en la posición correspondiente a esa palabra. Un ejemplo es el de la Ilustración 18.

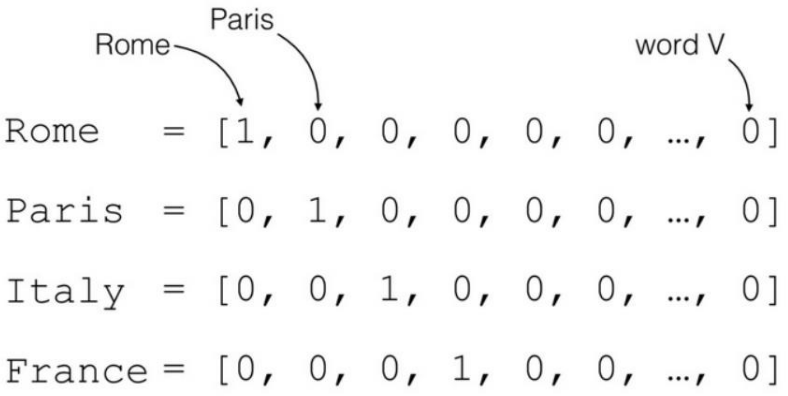


Ilustración 18: ejemplo codificación one-hot [14]

De esta forma para las frases que introduciremos se compondrán de matrices cuyos elementos serán los one-hot representando las palabras como se muestra en Ilustración 19.

Ilustración 19: representación frases codificación one-hot. [elaboración propia]

Las demás consideraciones y tratamiento del dataset en este proyecto se explican más adelante en Tokenización obra.

2.4 CREACIÓN MODELO

Para la creación del modelo se hará uso de las librerías de Python de Tensorflow y Keras. La API de estas librerías nos permite una vez entendido el funcionamiento de las redes neuronales y cuál es la estructura neuronal que queremos utilizar, crear las redes neuronales a un nivel más alto, controlando todos los parámetros, pero sin necesidad de programar toda la estructura de la red neuronal.

De esta forma con la LSTM layer de keras crearemos la red neuronal LSTM con la estructura explicada en Redes neuronales y LSTM. En este comando de Keras podremos controlar el número de neuronas de la red LSTM. También nos permite añadir varias capas a la red neuronal, uniendo por ejemplo dos redes neuronales consecutivas.

Otro de los parámetros que tendremos control será el tamaño de embedding, que es esencialmente una matriz de tamaño filas el número de palabras de nuestro diccionario y columnas el tamaño del embedding. Esta matriz de embedding se entrenará durante el proceso de entrenamiento, creando un vector para cada palabra relacionando todas las palabras del diccionario entre ellas de formando un mapa de dimensiones igual al tamaño del vector de embedding en el que las palabras más parecidas conceptualmente estén más cercanas.

Se muestra en la figura matriz embedding Ilustración 20 lo que es una capa embedding:

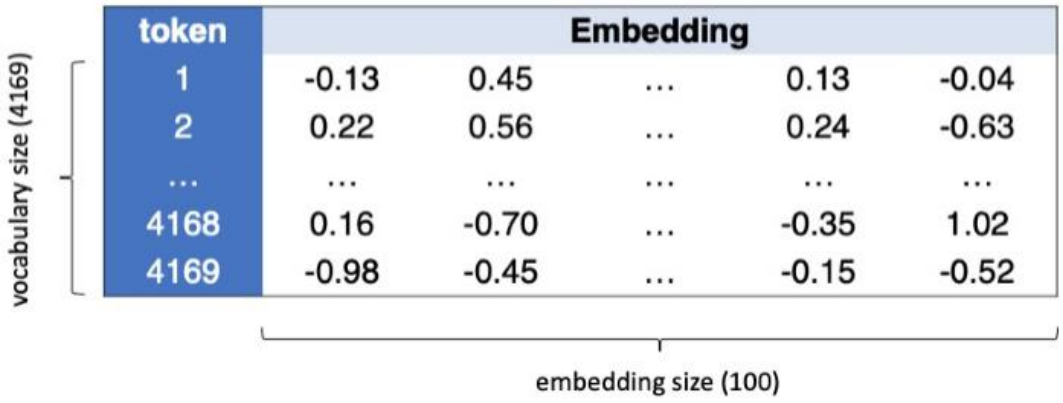


Ilustración 20: matriz embedding[17]

En la figura Ilustración 21 un mapa creado con el embedding de dos dimensiones del proyecto de referencia [19] en el que apreciamos como los sentimientos positivos son cercanos entre ellos y lejanos a los negativos.

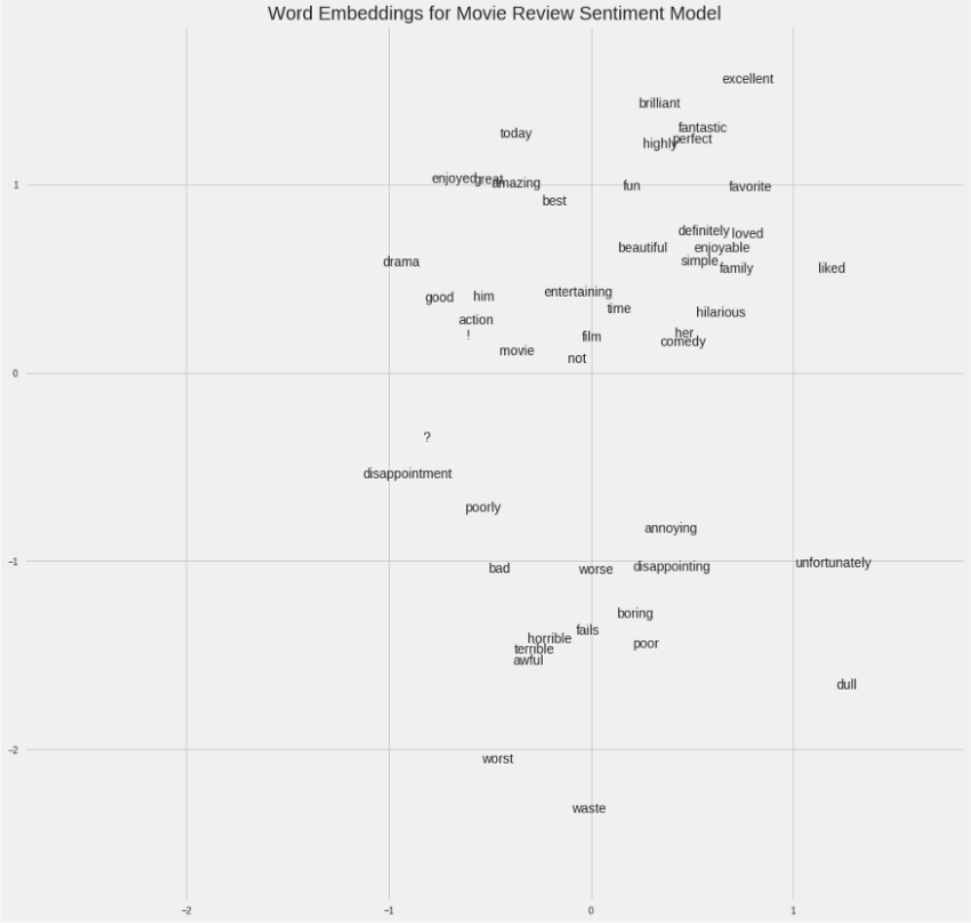


Ilustración 21: representación embedding sentimientos [18]

2.5 *ENTRENAMIENTO MODELO*

Para el entrenamiento también se hará uso de la API de keras haciendo uso de model.compile y model.fit.

Con model.compile definiremos la función de perdidas **Loss**, función que se encarga de computar el valor que debe minimizar el modelo en el entrenamiento. Existen distintos de tipos de funciones según como deseemos realizar el proceso de optimización. Las funciones de perdida pirobalísticas, las de regresión, y las de clasificación por margen máximo. [20]

Nuestro modelo utilizara la función de perdidas probabilística, defina en Keras como Categorical Crossentropy, útil para la codificación one-hot y que se define con la Ecuación 8. [22]

Ecuación 8: función de perdidas categorical crossentropy

$$\text{Loss} = -\sum_{i=1}^{\text{output size}} y_i \cdot \log \hat{y}_i$$

En la que y_i son las posiciones del vector objetivo, es decir el valor que nos gustaría que sacase el modelo si puede ser (podrá ser 0 si no es esa la posición o 1 si esa es la posición) y que conocemos porque forma parte del dataset. $\hat{y}_i$ es el vector con las probabilidades de cada posición que obtenemos del modelo. De esta forma menor sea la probabilidad de que se dé el resultado real, mayor es la perdida y viceversa.

Con model.compile también definiremos el **optimizador** de la red neuronal. El entrenamiento de la red neuronal es en realidad un problema de optimización pues el objetivo es minimizar la función de perdidas. Debemos tener en cuenta que las redes neuronales por su complejidad son problemas de optimización no convexos, por lo que existen mínimos locales además del mínimo local máximo ideal que buscamos.

Para hacer frente al problema de optimización existen diferentes métodos como puedes ser el Adadelta, el Adam o el Adagrad entre otros. En nuestro proyecto se utilizará el RMSprop. Definidas con las fromulas de Ecuación 9 y Ecuación 10:

Ecuación 9: gradiente RMSprop

$$v_{dw} = rho \cdot v_{dw} + (1 - rho) \cdot dw^2$$

Ecuación 10: pesos RMSprop

$$w = w - Lr \cdot \frac{dw}{\sqrt{v_{dw}} + \epsilon}$$

En Ecuación 9 se calcula una media exponencial del cuadrado del gradiente. Esta operación se hace para cada peso, y por ello se incluye el parámetro rho, que multiplicamos por la media exponencial calculada hasta el momento. Rho se define en Keras como el factor de reducción del historial del gradiente.[23]

Observamos que a partir de esta ecuación actualizamos los pesos en Ecuación 10. Donde observamos Lr el learning rate, que cuanto mayor sea mayor será la variación de los pesos en cada iteración. También observamos la presencia del parámetro épsilon cuya razón de ser es la posibilidad de que la media del gradiente puede ser muy próxima a 0 provocando que los pesos exploten.

Por lo tanto, los paramentros que nos permitirá cambiar Keras en el RMSprop serán rho, el learning rate y épsilon. Siendo los más interesantes rho y lr.

El **learning rate** será especialmente importante para la optimización correcta del modelo en un intervalo de tiempo aceptable. Si tenemos un learning rate muy pequeño, los pesos no cambiaran y nos acercaremos de forma demasiado lenta al mínimo. Por otro lado, si el learning rate es demasiado grande, los pesos cambiaran demasiado y daremos saltos demasiado grandes en la función de perdidas, por lo que acabaremos dando saltos que no tienen por qué estar dirigidos al mínimo. Esto puede verse representado en la figura Ilustración 22

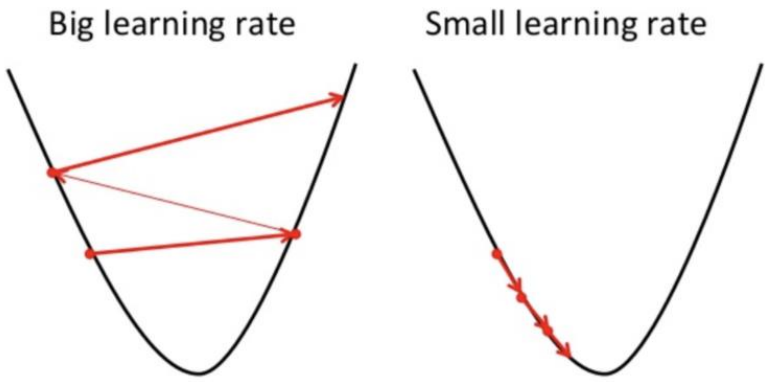


Ilustración 22: representación distintos tamaños learning rate[24]

Finalmente, con el model.fit entrenaremos al modelo indicando a keras los inputs del dataset y el vector objetivo de resultados del modelo.

En el model.fit también indicaremos el **batch size**, que indicara el número de muestras que son procesadas antes de actualizar los pesos del modelo y las **épocas**, que indica el número de veces que recorreremos el dataset.[29]

2.6 API TWITTER

Twitter pone a nuestra disposición la herramienta Twitter developer que nos permite solicitar una API key, un identificador que nos permitirá crear una app ligada a un usuario de Twitter, y con la que tendremos acceso a una serie de comando de la API pública de Twitter.

Una vez obtenida la API key, solo debemos autentificarnos, y con la ayuda de la librería de Python tweepy [32], podremos twittear, retwittear entre otros desde Python. La lista de acciones permitidas por la API se encuentra en el enlace de referencia [33] .

Capítulo 3. ESTADO DE LA CUESTIÓN

En noviembre de 2019, OpenAI, la compañía de inteligencia artificial fundada por Elon Musk, publicaba la versión completa de su modelo de lenguaje no supervisado GPT-2 [35], un modelo con una capacidad de generación de texto muy parecido a un texto humano. El 11 junio de 2020, se hizo disponible la beta privada de GPT-3. [36]

Meses después de la publicación completa de GPT-2, el 4 marzo de 2020, el medio de información web estadounidense, Vox, publicaba un pequeño reportaje “Computers just got a lot betthet at writing” [37], en el que demostraba la reciente evolución de la generación de texto por inteligencias artificiales. Entre los proyectos referenciados por Vox, muchos hacen uso de GPT-2.

Entre estos proyectos encontramos uno realizado por el Allen Institute for AI, que a partir del modelo GPT-2, nos va diciendo cual son las siguientes palabras más probables dada una frase o texto. [38]

Otro de los proyectos referenciados realizados con GPT-2 es un detector de la probabilidad de que un texto sea real o escrito por una inteligencia artificial [39]. Al introducir por ejemplo la frase: “Carlos ‘s final degree poject grade will be greater than eight.” el modelo considera que la probabilidad de que esta frase sea real es 53,15 %, pero no puede confirmarse nada, pues para que sea fiable debe introducirse al menos 50 tokens. Como observamos en la Ilustración 23.

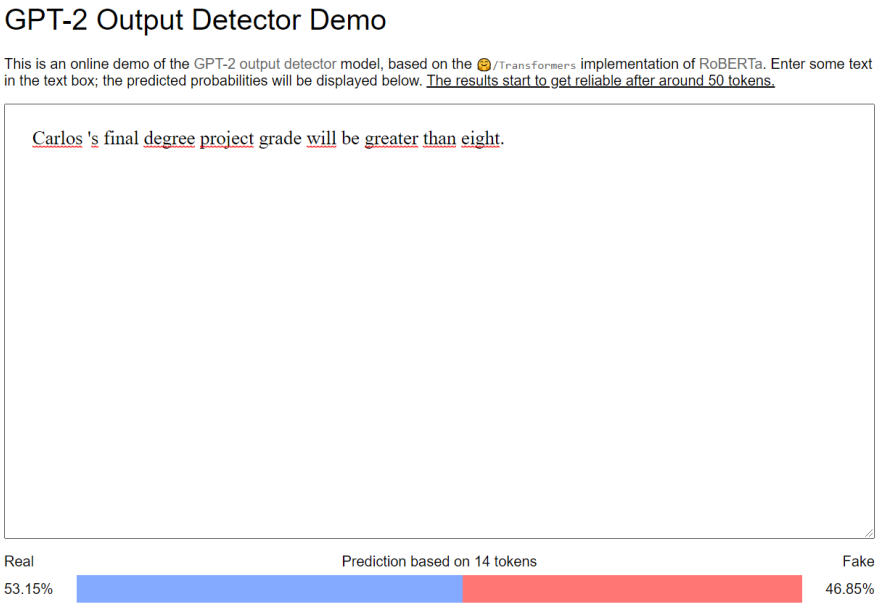


Ilustración 23: ejemplo detección texto artificial[39]

Pero no solo se referenciaban proyectos con GPT-2. Existen otros modelos como puede ser Grover desarrollado por el Allen Institute for AI, capaz de realizar artículos de periódico "Fake news" [40].

Otro ejemplo es el modelo Haim, desarrollado por AI21labs, que genera texto a partir de un texto inicial proporcionado [41]. Por ejemplo, al proporcionar a la IA el texto:

"I was writing my final degree project when"

Haim devuelve:

"I was writing my final degree project when I found out the news. I started by writing a quick blog post explaining the situation. I was already feeling bad, but I did it anyway because I was sick of being a geek. The thing is, no one at MS has heard from the family. Or they have been informed."

También se hacía referencia otras propuestas más originales como AIdungeon.io, un juego de mazmorras con posibilidades infinitas, en el que podemos introducir en forma de texto

la siguiente acción que queremos realizar y el juego creara el escenario y la historia según lo que vayamos introduciendo [42].

Viendo la gran cantidad de proyectos recientes dirigidos a la computación del lenguaje, observamos como este concepto está en plena etapa de evolución y desarrollo.

Sin embargo, en cuanto nos centramos en poesía, los proyectos escasean, aunque existen. No es tan fácil encontrar proyectos centrados en este aspecto del lenguaje, en los que se debe analizar el lirismo, figuras literarias y metáforas en las que las palabras a veces no tienen su significado literal, añadiendo dificultad.

Uno de los proyectos de mayor magnitud es Poem Portraits [43], desarrollado por Google, y en el que aportamos una imagen y una foto para obtener un poema personalizado. Por ejemplo, al aportar la palabra "engineering" y una foto obtenemos Ilustración 24:

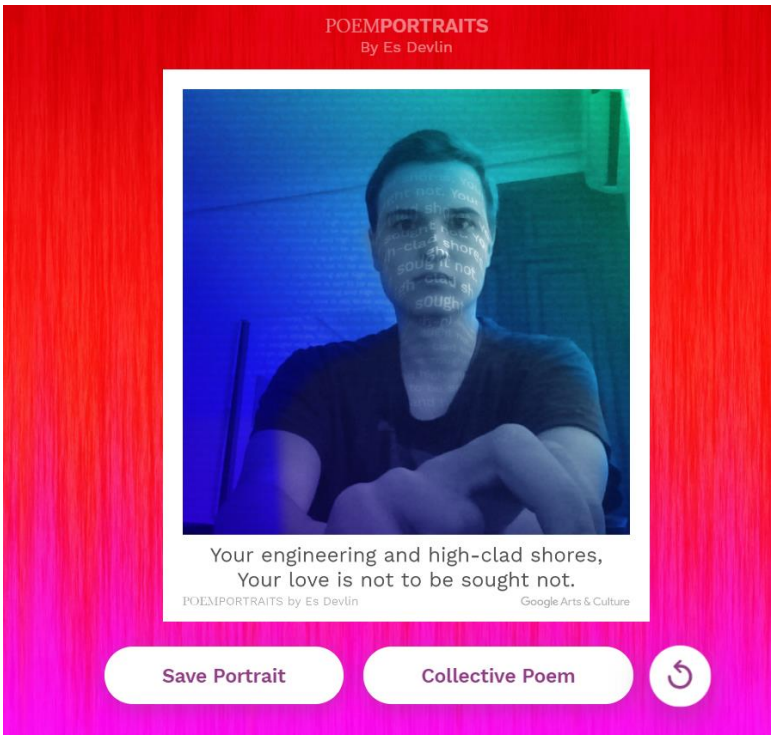


Ilustración 24: Ejemplo Poem Portraits[43]

Una característica común de todos los modelos mencionados hasta ahora y que difiere con nuestro proyecto es que los modelos hablan inglés. Pues a pesar de que el español es el segundo idioma más hablado tras el chino y una de las elecciones preferidas de aquellos que desean aprender un nuevo idioma, las inteligencias artificiales no parecen tener tanto interés en aprenderlo. De la misma forma en la que el inglés por su facilidad de aprendizaje y sencillez se convirtió en el idioma oficial de negocios y relaciones internacionales, parece estar convirtiéndose en el idioma oficial de la inteligencia artificial.

Las grandes empresas punteras en inteligencia artificial son en su mayoría estadounidense, y si bien China está realizando un gran esfuerzo por ponerse al día, la UE se está quedando atrás. [44]

Eso no quita que, aunque a causa de esto los proyectos no tienen la misma magnitud, existan múltiples proyectos centrados en la generación de texto con inteligencias artificiales en castellano. Incluso haciendo uso del gran generador de texto en ingles el GPT-2 como es el proyecto de un chat-bot en español de referencia [45] y que demostró que se puede usar el modelo para otros idiomas que no sea inglés.

En cuanto a la poesía, existe el proyecto WASP (the Wishful Automatic Spanish Poet)[46], que lleva alrededor de 20 años en continua evolución y desarrollado por el catedrático de la Universidad Complutense de Madrid Pablo Gervás [47]. El proyecto lleva años investigando sobre la computación de la poesía en castellano y la creación de inteligencias artificiales dirigidas a este ámbito.

Finalmente, introduciremos nuestra inteligencia artificial en Twitter creando un bot poeta. Existen algunos bots parecidos en Twitter, pero cabe sobre todo destacar el bot del usuario @BotPoeta de referencia [49] y programado por Eduardo Geûens. El Bot no hace uso de la inteligencia artificial y está programado directamente a través de reglas gramaticales y métricas, pero es sin duda parecido al robot que se planteara en este proyecto, creando un monopolio que podría incluso considerarse desleal por la familia Geûens.

Capítulo 4. DEFINICIÓN DEL TRABAJO

4.1 JUSTIFICACIÓN

En el Estado de la Cuestión se ha hablado de proyectos similares a los de este documento, por ello ha de plantarse la importancia de nuestro bot poeta y por qué es apropiado realizarlo.

Como hemos visto, en todos los proyectos presentados en el capítulo anterior, se trata de proyectos recientes y en ocasiones todavía en desarrollo. La inteligencia artificial y el tratamiento computacional de texto está en pleno progreso y crecimiento. Por ello cualquier proyecto de investigación ahora mismo es necesario para seguir impulsándolo a nuevas cuotas e investigar sus límites.

Esta podría ser la primera justificación de este proyecto, un ejercicio teórico buscando las limitaciones de la LSTM, y ver si esta es capaza de comprender algo tan creativo y humano como es la poesía. Buscar hasta qué punto nuestra red neuronal puede convertirse en un ente creativo, que sea capaz no solo de generar texto, si no texto lírico. En definitiva, estudiar y plantearnos si las inteligencias artificiales pueden crear arte.

Otra justificación, no menos importante, es que es divertido. No solo debe pensarse en aplicaciones pragmáticas de la inteligencia artificial. Hay algo fascinante y entretenido en ver inteligencias artificiales realizar actos creativos y humanos. Pero no es únicamente válida esta justificación, por el valor de la diversión, tan válida como cualquier razón para una investigación, sino que es la forma de seguir impulsando el desarrollo. No hay estudio e investigación más eficaz que aquella que se hace por verdadero interés y diversión. No solo eso, sino que llevara más gente a la indagación en el tema. Desde el año 2000 el porcentaje de alumnos en carreras de STEM en España se ha reducido en un 30%[50]. En un mundo más tecnológico que nunca, alumnos enganchados a la tecnología e internet, deciden no optar por carreras tecnológicas porque “no compensa” o falta de interés. El alumno que finalmente se decida por una carrera STEM probablemente sea por aplicaciones más amenas de las tecnologías en desarrollo y no por sus aplicaciones más prácticas.

Por otro lado, para plantearse si una inteligencia artificial es capaz de hacer poesía, también debemos entender que es poesía. La cuestión va en los dos sentidos y preguntarse si una red neuronal puede escribir poesía, equivale a preguntarse si la poesía puede escribirse por algo que no sea un humano. Una vez obtenida la red neuronal, se analizarán los versos finales de este proyecto para ver si parecen poesía. Pero luego ha de plantearse si es poesía algo escrito por una inteligencia que entiende las palabras como números dentro de un problema computacional y sin sentimiento alguno.

Se trata por lo tanto no únicamente de un ejercicio teórico sobre redes neuronales, sino también de un ejercicio poético. Si realmente decidimos que aquello que se obtendrá de la red neuronal es arte, la justificación del proyecto, es la razón por la que escribimos poesía. Esta pregunta responde Robin Williams en el Club de los poetas muertos (1989) y dice: "Leemos y escribimos poesía porque somos miembros de la raza humana, que desborda de pasión […] la poesía, la belleza, el romance, el amor, son las razones por las que nos mantenemos vivos". Pero más interesante todavía es otra frase de este mismo discurso en la que dice: "La medicina, el derecho, los negocios, la ingeniería son carreras nobles y necesarias para mantener la vida" implicando que la ingeniería carece de la pasión de la poesía y eso no es cierto. Como la poesía, la ingería es creativa, como el poeta pone su alma en su obra, puede hacerlo un ingeniero en su proyecto, y ambos si realizados correctamente afectaran significativamente a quienes van dirigidas. Es un error definir y categorizar las carreras y campos del conocimiento, que son complejos y puede están entrelazados entre sí. Al final de la escena, el personaje de Robin Williams cita al poeta Walt Withman que define y da sentido a la existencia diciendo que: "[…]puedes contribuir con un verso" y pregunta a sus alumnos: "Cual será tu verso?". Este proyecto, es la justificación más literal de qué ese verso puede obtenerse a través de una red neuronal, de qué ese verso puede ser la ingeniería.

4.2 OBJETIVOS

Un primer objetivo será crear y desarrollar una red neuronal que sea capaz de crear algo parecido a poesía.

La intención será a partir de un poeta elegido, ser capaz de recrear su poesía, con una inteligencia artificial creativa. Por tanto, las características que se intentará que tenga el modelo, será la creación de versos únicos y originales. Se buscará por tanto que no reproduzca exactamente la obra del autor, sino que sea capaz de crear versos nuevos.

En cuanto a la calidad de los versos que se intentara obtener, se buscara que le modelo llegue a un punto en el que apreciemos que tiene cierto conocimiento gramatical, diferenciando tipos de palabras, sustantivos, verbos, adverbios, adjetivos… y siendo capaz de tener cierta estructura. Se tendrá en cuenta la dificultad de obtener una sintaxis y gramática perfecta, siendo especialmente difícil obtener las conjugaciones requeridas, y la coincidencia de plurales y singulares de artículos y sustantivos, necesitándose un dataset de muy gran dimensión para cubrir todas las minucias del léxico español, pero se hará todo lo viable para alcanzar los mejores resultados posibles.

También se intentará que los resultados tengan el mayor valor poético que se pueda obtener, la intención no es únicamente que el modelo sea capaz de obtener un texto coherente, sino que sea poesía. Por ello se intentará en medida de lo posible obtener versos con valor artístico y que se pueda pensar que hayan sido escritos por un poeta humano.

Si se alcanza este primer gran objetivo, se tendrá como segundo objetivo su implementación dentro de otros códigos para a partir del modelo creado realizar acciones y aplicar el modelo a distintas funciones.

Una de las implementaciones del modelo que se intentaran será la realización de un bot de Twitter que utilice el modelo para tuitear poesía. Otra de las implementaciones será crear obras literarias a partir del modelo.

En definitiva, el objetivo es estudiar las redes neuronales LSTM, su capacidad para aprender y generar poesía y su implementación en códigos sencillos.

4.3 METODOLOGÍA

Para alcanzar los objetivos planteados, se realizará primero un trabajo de investigación y aprendizaje. Con ello se pretenderá hacer una primera aproximación a la inteligencia artificial y realizar una profundización en las redes neuronales para poder afrontar el proyecto de forma correcta. Este proceso de aprendizaje se realizar a través de cursos online, como Coursera[52], y a través de documentos sobre el tema y libros como "Generative Deep learning, teaching machines to write, compose and paint"de David Foster [17].

Una vez superada la primera etapa de familiarización con las redes neuronales, se pasará a la creación y entrenamiento de la red neuronal con Python y las herramientas que dispone Keras sobre redes neuronales.

Se realizará un proceso de selección del autor y procesado de la obra y se afrontará el proceso de creación y entrenamiento del modelo, mediante un proceso de análisis y estudio de aquello que vallamos obteniendo. Para la programación inicial en Keras se estudiarán proyectos parecidos con programaciones similares como el proyecto propuesto por David Foster en su libro "Generative Deep learning, teaching machines to write, compose and paint". Una vez realizada esta primerea aproximación estudiaremos que podemos mejorar y aportar al código.

Finalmente, si se consigue crear una red neuronal que cumpla los objetivos propuestos se pasara a crear el Bot con la API de Twitter.

Capítulo 5. SISTEMA/MODELO DESARROLLADO

A continuación, se muestra la realización del proyecto con los conceptos explicados Descripción de las Tecnologías con la intención de realizar los objetivos propuestos en Objetivos.

5.1 TOKENIZACIÓN OBRA

5.1.1 OBRA Y DICCIONARIO

El primer paso del proyecto se centró en la búsqueda de una obra apropiada que pudiera valer como conjunto de entrenamiento del proyecto. Se empezó entonces una búsqueda de un autor español con una obra que cumpliera una serie de condiciones para formar un dateset válido.

Era importante que el dataset elegido tenga cierto tamaño y tengamos una obra de una longitud suficiente para que al realizar el entrenamiento el modelo tenga información suficiente para aprender correctamente. También fue importante buscar una obra en un formato que pudiéramos tratar correctamente y modificar fácilmente con Python.

Teniendo estas consideraciones en cuenta se optó finalmente por usar la obra de Vicente Aleixandre obtenida https://www.poesi.as/Vicente_Aleixandre.htm. Se transfirió la obra a un txt para su tokenización y tratamiento en Python fácilmente.

Una vez decidida la obra, se necesitaba un diccionario para la tokenización de la obra.

En primera instancia se decidió buscar un diccionario castellano ya existente a partir del cual toekenizar la obra. Se buscó en repositorios como github.com, con la intención de encontrar un diccionario de uso sencillo y que nos facilitara la tarea. Se encontraron varios diccionarios y se pasó a la creación de secuencias como se explica en Creación secuencias, sustituyendo las palabras por un numero con su posición en el diccionario.

Como no todas las palabras de la obra tenían por que encontrarse dentro del diccionario seleccionado se decidió añadir una palabra en la última posición "unknow" para asignarla a aquellas palabras que no se encontraran en el diccionario. Tras realizar las secuencias con este método se comprobó que era un método errónea y poco eficaz. Mientras la obra contenía verbos en todas sus conjugaciones, palabras en todas sus variantes, masculino, femenino, plural, singular… El diccionario seleccionado solo tenía los verbos en infinitivo y no tenía la mayoría de las palabras de la obra. A causa de esto nos encontramos con unas secuencias con una gran mayoría de "unknows" que provocaría un entrenamiento muy pobre del modelo.

Al darnos cuenta de este error se podría haber optado por dos opciones, una primera habría sido buscar un diccionario más completo, pensado para este tipo de problema, en el que se encontrase un mayor de variedad de palabras en todas sus formas. Estos diccionarios existen y pueden encontrarse, incluso con embeddings ya entrenados de red neuronales. Sin embargo, el tamaño de estos diccionarios suele ser inmenso, lo que provoca que se muy pesado y lento trabajar con ellos y siguen sin asegurar que no tengamos "unknows".

La segunda opción, más coherente y por la que se optó fue crear un diccionario a partir de la obra. Es decir, ir leyendo cada una de las palabras de la obra y guardándolas en un diccionario de creación propia que utilizaremos como índices para las secuencias, asegurándonos de esta forma tener en el diccionario todas las palabras de la obra y no tener palabras innecesarias que solo aumentarían el tamaño del diccionario sin ser de utilidad.

De esta forma se creó un código en Python que abriera el txt con la obra de Vicente Aliexandre e hicimos uso del comando split, que nos permite separar los caracteres de un txt a partir de los caracteres deseados. Indicando con nos separa el txt por los espacios y todos los puntos de puntuación guardamos todas las palabras de la obra en una lista. Para de esta forma en un bucle ir leyendo cada palabra y si no ha sido leída anteriormente escribirla en un txt que nos servirá como diccionario. En este txt que utilizaremos como diccionario también se encontraran las etiquetas "sof" para indicar el inicio de cada verso y "eof" para indicar el final de cada verso.

Todo el código de creación del diccionario se muestra en el ANEXO I: .

5.1.2 CREACIÓN SECUENCIAS

Una vez creado el diccionario podemos pasar a la creación de secuencias y la transformación de la obra en algo que nuestro modelo pueda comprender.

Para ello lo primero es pensar cómo se quiere introducir a nuestro modelo el dataset y como se quiere que aprenda. Para ello debemos tener claro cuáles serán los vectores objetivo es decir que outputs queremos que el modelo aprenda a realizar y como queremos que el modelo entienda estos outputs. Es decir, no debemos únicamente traducir el dataset a secuencias binarias que podemos introducir en el modelo, debemos modificarlo para que el modelo entienda el dataset como queremos.

En una instancia inicial, se decidió entender el dataset como un conjunto de secuencias con inicio y un final, formada por cada verso. Es decir, decidimos enseñar al modelo a realizar versos, introduciéndolos como una secuencia de palabras con una dependencia recurrente con un principio y un final. Para ello, se modificó nuestro txt original con Python, introduciendo al principio de cada línea un "sof" como inicio de verso y al final de cada línea antes del salto de línea "/n" un final de verso "eof", introduciendo estas dos palabras en el diccionario para proceder a la secuenciación.

En Ilustración 25podemos ver una estrofa inicial de dataset.

```
La tristeza u hoyo en la tierra,
dulcemente cavado a fuerza de palabra,
a fuerza de pensar en el mar,
```

Ilustración 25: estrofa inicial dataset [elaboración propia]

En la Ilustración 26 podemos ver una estrofa con el inicio y fin de verso añadido.

```
sof La tristeza u hoyo en la tierra, eof
sof dulcemente cavado a fuerza de palabra, eof
sof a fuerza de pensar en el mar, eof
```

Ilustración 26: estrofa con inicio y fin de verso [elaboración propia]

Como se explicó en el capítulo Tokenización, se utilizó una codificación one-hot en la que cada palabra corresponde a un vector del tamaño del diccionario de 0 con un 1 en la posición de la palabra. Sin embargo, para reducir el tamaño del txt de secuencias se decidió crear inicialmente un txt con las secuencias en el que cada palabra es equivalente únicamente al número de su posición en el diccionario.

Una vez decidido como vamos a crear las secuencias, creamos el código en Python para realizar este proceso. Como vamos a tratar el dataset como secuencias de versos, separamos a los versos con split('eof'), que hemos introducido como final de versos para tener todos los veros separados. Una vez tenemos los versos separados creamos una función renderVerso, que se separa todas las palabras del verso como hicimos para la creación del diccionario. Una vez separadas las palabras la comparamos en una función con todas las palabras del diccionario, hasta encontrar la palabra del diccionario con la coincide y guardamos la posición, creando un vector de verso renderizado en el que las palabras se sustituyen por su número de posición en el diccionario. Una vez renderizado vamos escribiendo las secuencias en un txt para al final del proceso tener un txt con todas las secuencias. Todo este código se encuentra completo en ANEXO II.

De forma que siendo el diccionario para esta primera estrofa que hemos estado analizando el de Ilustración 27

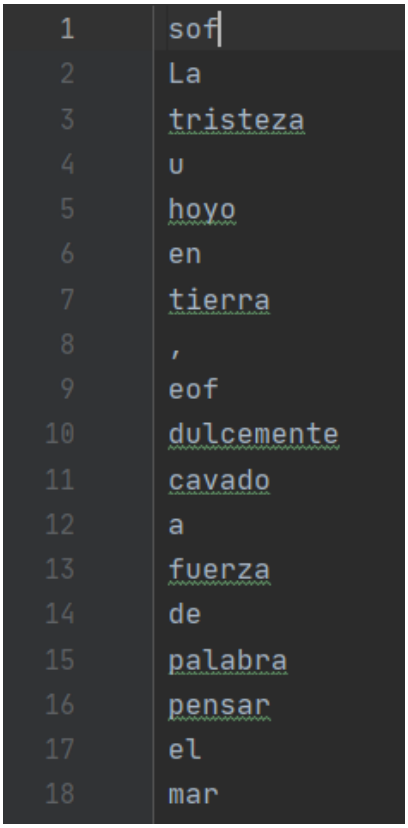


Ilustración 27: diccionario primera estrofa [elaboración propia]

Obtendríamos la secuencia de la Ilustración 28

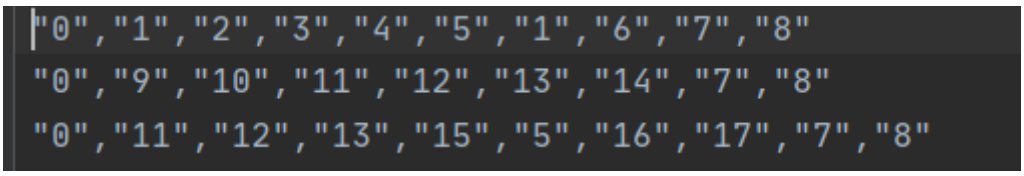


Ilustración 28: ejemplo secuencias primera estrofa [elaboración propia]

Con esto ya tendríamos unas secuencias válidas para trabajar en el proyecto y esto fue lo que se utilizó durante gran parte del mismo. Sin embargo, una vez demostrado que esto funcionaba y que le modelo funcionaba correctamente se decidió ir un paso más allá y no solo tratar el dataset como versos, pero también como estrofas.

Debido al que el dataset tiene una línea en blanco entre estrofa y estrofa, el procedimiento fue bastante sencillo en cada línea en blanco se añadió un final de estrofa “eop” y al principio

de cada estrofa un inicio de estrofa "sop". Una vez tenemos preparado el dataset de esta forma realizamos el mismo proceso que para los versos, pero separando el dataset por los "eop" en lugar de los "eof" y seguimos el mismo proceso para realizar las secuencias. El código completo para realizar esto se muestra en ANEXO III.

De forma que una estrofa del dataset quedaría de la siguiente manera como se muestra en Ilustración 29: estrofa con inicio y final de estrofa.

```
sop sof ¿Qué firme arquitectura se levanta eof
sof del paisaje, si urgente de belleza, eof
sof ordenada, y penetra en la certeza eof
sof del aire, sin furor y la suplanta? eof
eop
```

Ilustración 29: estrofa con inicio y final de estrofa [elaboración propia]

El siguiente paso es obvio y es tratar las poesías enteras como secuencias. Esto sería más trabajoso pues para indicar en el dataset las diferentes poesías deberíamos introducir el final e inicio de poesía a mano o encontrar un dataset en el que de alguna forma se diferenciara entre las poesías para automatizar le proceso. Esto no se llegó a realizar en el proyecto.

5.1.3 SECUENCIAS EN EL MODELO

Una vez tenemos las secuencias hay que tener claro cuál será el conjunto de entrenamiento y cual el vector objetivo que deseamos que el modelo intente acercarse reduciendo el error.

Inicialmente cogiendo inspiración del modelo propuesto por David Foster en su libro "Generative Deep learning, teaching machines to write, compose and paint" [17], se realizó una primera propuesta. Se dividió el conjunto de entrenamiento en secuencias de la misma longitud y se hizo que el vector objetivo fuera la siguiente palabra de estas secuencias.

Se entrenó de esta forma al modelo y se intentó optimizar la parametrización obteniendo unos resultados con una estructura similar a lo que podría considerarse un verso. Sin

embargo, no se apreciaba demasiada estructura gramatical y en general los resultados fueron peores de lo que se esperaba obtener.

Por ello se reanalizó el modelo y se decidió cambiar las secuencias de entrenamiento y el vector objetivo. Se crearon secuencias de versos del tamaño del verso más largo, alargando el final de verso hasta el tamaño de las secuencias. Es decir, si el verso más largo resulto ser de por ejemplo 25 palabras, todas las secuencias son de 25 palabras y si uno de los versos es de por ejemplo 10 palabras se completará hasta 25 con 15 "eof".

Finalmente, se cambio el vector objetivo no únicamente a una única palabra sino al verso completo sin la etiqueta de inicio, en formato one-hot explicado en Tokenización. Es decir, forzamos al modelo a crear versos enteros, creando la dependencia secuencial de todo el verso. Este código puede verse en el ANEXO IV.

Una vez realizado este cambio, los resultados fueron notablemente mejores.

5.2 MODELO CON KERAS

5.2.1 CREACIÓN MODELO

Tras la secuenciación del dataset, nos centramos en la creación de la red neuronal en keras.

Para ello primero definimos la capa embedding que será nuestra capa de entrada al modelo. Con estas capas sustituimos cada uno de los índices del diccionario por un vector del tamaño el embedding como se explicó en Creación modelo. Escribimos entonces la siguiente línea de código:

```
x = Embedding(total_words, embedding_size)(text_in)
```

En ella indicamos el número de palabras y el tamaño que hayamos decidido del embeddding, aunque no sabemos exactamente que numero será el más apropiado y habrá que optimizar.

Una vez tenemos esta primera capa de entrada se pasa a la red neuronal LSTM que hemos escogido por las razones explicadas en Redes neuronales y LSTM y que tendrá todas las

características explicadas con las funciones de activación por defecto de la LSTM, tanh y sigmoid. Añadimos por lo tanto la LSTM con la línea de código.

```
x = LSTM(n_units, dropout=0.2, recurrent_dropout=0.2, return_sequences=Tr
ue)(x)
```

En este codigo n_units nos indica la dimensión de la capa de salida de la LSTM. No sabemos que numero n_units será el más apropiado y que consiga que la rede recurrente funcione y realice lo que queremos de la forma más optima.

El dropout será siempre un numero entre 0 y 1 que se refiere al porcentaje de neuronas que ignoraremos durante la transformación lineal del input. Es decir, neuronas que no utilizaremos con una probabilidad definida para el tratamiento del input en la LSTM. Esto se realiza para evitar el sobre entrenamiento. Estas neuronas "ignoradas" se representan el la Ilustración 30.

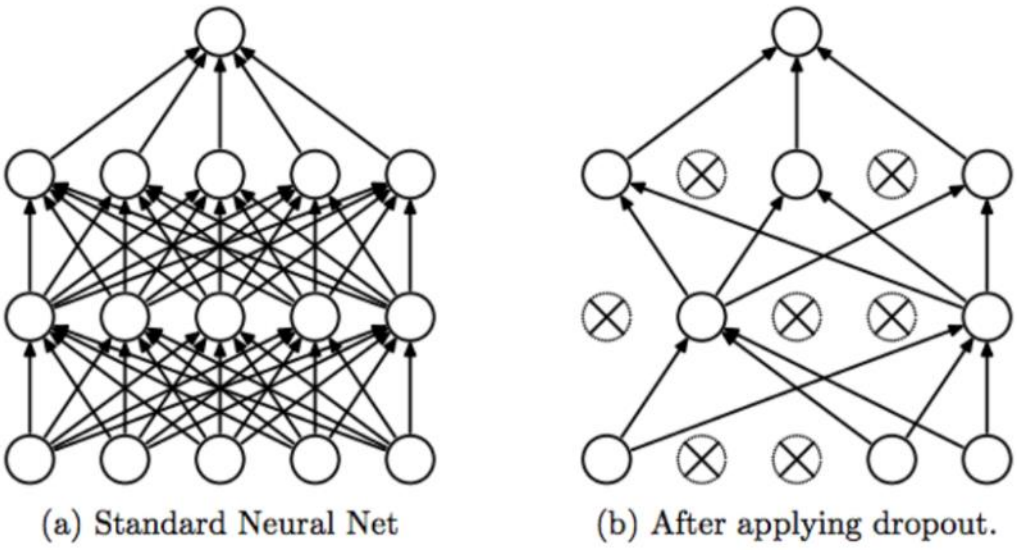


Ilustración 30: representación dropout en una red neuronal[55]

El recurrent dropout cumplirá la misma función, pero para el estado recurrente en lugar de los inputs.

Por último, ponemos return_sequences a True, para que la red devuelva el ultimo output obtenido.

Finalmente, añadiremos la capa de salida con Dense, que convertirá la salida de la LSTM en un vector de probabilidades de cada palabra del tamaño del diccionario. Esto se realizar con la función de activación softmax explicado en Redes neuronales y LSTM. Nos dará por tanto la salida del modelo, con las probabilidades del siguiente output una vez entrenada la red neuronal y que podremos utilizar para crear los versos. Todo esto se realiza con la línea de código:

```
text_out = Dense(total_words, activation='softmax')(x)
```

5.2.2 ENTRENAMIENTO MODELO

Una vez creada la red neuronal y tiendo las secuencias que queremos que el modelo aprenda solo nos queda entrenar a la red con el razonamiento explicado en Entrenamiento modelo.

Para ello lo primero que se hizo es definir el optimizador a usar con el lr que se desee, en nuestro caso el RMSprop y se definio la función de pérdidas que el modelo intentará minimizar al entrenar el modelo, en nuestro caso categorical crossentropy, ambos explicados en Entrenamiento modelo.

Todo esto se hizo con el comando model.compile de keras con las siguientes líneas de código:

```
opti = RMSprop(lr=0.001)

model.compile(loss='categorical_crossentropy', optimizer=opti)
```

Tras esto, se pasó al entrenamiento del modelo, indicando cual es el vector obejtivo definido en Secuencias en el modelo. Hay que indicar el numro de veces que se quiere recorrer el dataset y el numero de muestras procesadas cada vez que cambiemos los pesos del entrenamiento. Todo esto se realizo con model.fit en keras con la siguiente línea de código.

```
model.fit(X, y, epochs=epochs, batch_size=batch_size, shuffle=True)
```

Una vez, realizados estos pasos se puede empezar el entrenamiento y realizar el proceso computacional recorreindo el dataset tatnas veces como ahllamos indicado y cambiando los pesos reduciendo la función de perdidas. Este proceso se seguirá y supervisara a través de Python como se muestra en Ilustración 31.

```
  16/1700 [..............................] - ETA: 14:26 - loss: 9.2912
  32/1700 [..............................] - ETA: 8:11 - loss: 9.2532
  48/1700 [..............................] - ETA: 5:52 - loss: 8.7917
  64/1700 [>.............................] - ETA: 4:41 - loss: 8.2235
  80/1700 [>.............................] - ETA: 3:59 - loss: 7.6267
  96/1700 [>.............................] - ETA: 3:47 - loss: 6.9998
 112/1700 [>.............................] - ETA: 3:24 - loss: 6.4036
 128/1700 [=>............................] - ETA: 3:22 - loss: 5.8723
 144/1700 [=>............................] - ETA: 3:08 - loss: 5.4609
 160/1700 [=>............................] - ETA: 2:58 - loss: 5.0878
 176/1700 [==>...........................] - ETA: 2:52 - loss: 4.7452
 192/1700 [==>...........................] - ETA: 2:51 - loss: 4.4720
 208/1700 [==>...........................] - ETA: 2:41 - loss: 4.2223
 224/1700 [==>...........................] - ETA: 2:33 - loss: 4.0184
 240/1700 [===>..........................] - ETA: 2:26 - loss: 3.8576
 256/1700 [===>..........................] - ETA: 2:20 - loss: 3.6967
 272/1700 [===>..........................] - ETA: 2:15 - loss: 3.5598
 288/1700 [====>.........................] - ETA: 2:09 - loss: 3.4290
```

Ilustración 31: proceso entrenamiento visulaizado en Python [elaboración propia]

En la ilustración observamos, el proceso de entrenamiento de cada época, en la que podemos ver como avanza el proceso de secuencias analizadas que avanza según el batch size indicado. Se nos indica también el tiempo aproximado que queda del entrenamiento de la época y el error estimado por la función de activación en cada punto, que se ira reduciendo durante el proceso de entrenamiento. Como la intención es que la función de perdidas del entrenamiento sea lo máximo posible, necesitamos entrenar el modelo el numero de épocas necesario para que esto ocurra, por lo que durante el proceso de entrenamiento tendremos

que supervisar las perdidas en cada época para comprobar que no nos quedamos cortos ni nos pasasmos.

Una vez entrando el modelo solo quedara guardarlo con model.save para poder utilizarlo posteriormente.

Todo el código de la creación de la red neuronal y su entrenamiento se encuentra en ANEXO V.

5.2.3 GENERACIÓN TEXTO

Una vez guardado el modelo nos podremos centrar en la generación de texto.

Se tiene un modelo entrenado con secuencias que comienzan con inicio para que nos saque recurrentemente estas secuencias. Por lo que para generar texto se introduce al modelo como primera palabra la etiqueta correspondiente a inicio y creamos las secuencias a partir de ahí.

Obtendremos al dar el modelo esta primera palabra, las probabilidades de todas las palabras que elegiremos por lo tanto de forma estocástica en lugar de determinista, no eligiendo siempre la palabra con mayor probabilidad. Este carácter estocástico se realiza con una temperatura en una función de sampling que nos indicará como de determinista será la secuencia que nos entregue el modelo.

A partir de esta función de sampling se escogerá la siguiente palabra de la frase, que guardaremos con el resto de la frase que teníamos previamente para introducir nuevamente al modelo la frase con la nueva palabra y volver a samplear. Este proceso se realiza hasta que lleguemos a la etiqueta que nos indique el final de verso o estrofa y que significa que hemos llegado al final de la secuencia.

El código para realizar esto se encuentra en ANEXO VI.

5.3 IMPLEMENTACIÓN MODELO

Se tiene por lo tanto un modelo entrenado capaz de crear versos. Ahora existen infinitas posibilidades y formas en las que aplicar este modelo. En este proyecto se proponen dos formas un Bot de Twitter que vaya tuiteando cada cierto tiempo los versos y la otra un creador de obras literarias.

5.3.1 BOT TWITTER

Lo primero que debe realizarse es obtener un APIkey solicitando a Twitter para poder acceder a la API y poder programar nuestro Bot.

Aunque hace unos años el proceso era más sencillo y obtener una calve era muy simple, recientes polémicas sobre la privacidad de datos, el uso excesivo de bots para venta de productos o el intento de influenciar a la gente en ciertos temas, ha dificultado el proceso. Por eso lo primero que hay que hacer es aplicar para obtener una cuenta en Twitter developper.

Al aplicar para Twitter developper tendremos que explicar cuál es nuestra situación como desarrollador, en nuestro caso estudiante con un proyecto de investigación. Además, deberán darse una serie de explicaciones sobre cuál es la intención de los bots que vamos a programar cuáles son sus objetivos y que tipo de datos trataremos y analizaremos. Tras una serie de comprobaciones de identidad obtendremos acceso.

Una vez obtengamos acceso a Twitter developper, se nos dará acceso para crearar nuestra app dentro de Twitter. Como vemos en la interfaz de la Ilustración 32.

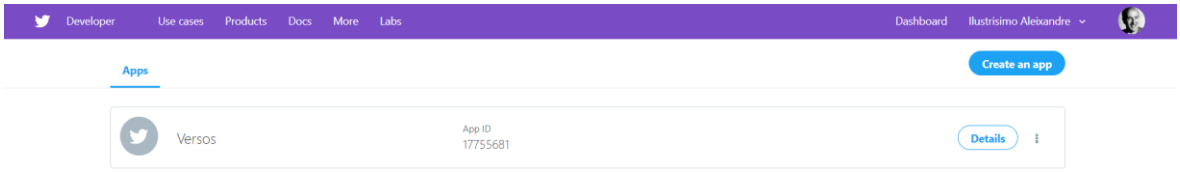


Ilustración 32: interfaz Twitter developper[30]

Al crear una nueva app también tendremos que definir cuál será su uso y darle una pequeña descripción. Finalmente obtendremos nuestra calve como se muestra en Ilustración 33 donde se ha borrado la clave por cuestiones de privacidad.

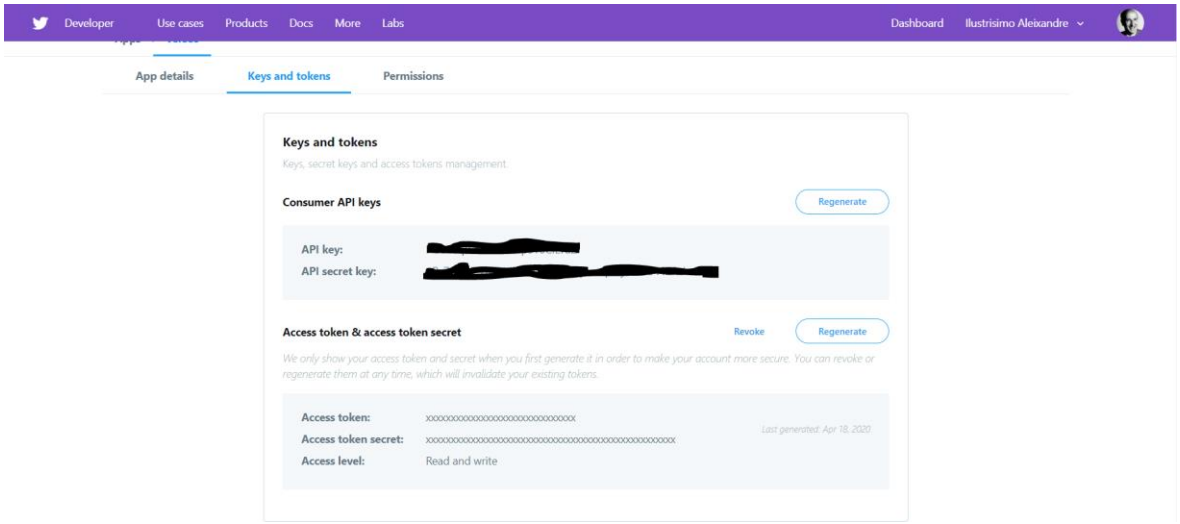


Ilustración 33: API key en Twitter developper[30]

Una vez obtenemos la clave API podemos consultar en [33], todos los comandos a los que tenemos acceso.

Ahora es turno de programar en Python el bot. Para ello hacemos uso de tweepy, que nos permite realizar una rápida y sencilla autentificación de la clave de la app, permitiéndonos empezar a programar las acciones que queremos que realice el bot.

Como la idea es tuitear un poema de nuestro modelo cada cierto tiempo, se carga en nuestro código de Python el modelo, y el script de generación de versos. Una vez cargado esto dentro de un bucle infinito esperaremos el tiempo seleccionado, y una vez esperado ese tiempo llamaremos a la función generate_text que nos generara una secuencia de versos. Una vez se obtenga la secuencia de versos se comprobará que su número de caracteres sea menor que 160, si no es así se pedirá otro verso al modelo. Una vez se obtenga un verso menor de 160 caracteres se tuitea haciendo uso de la API de Twitter. Una vez tuiteado, se volverá al principio del bucle para repetir el proceso hasta que se decida desconectar al bot.

Todo el código de este bot se muestra en el ANEXO VII.

5.3.2 CREACIÓN DE OBRAS

Otra de las aplicaciones donde se decidió introducir el modelo fue un creador de obras literarias. De forma que, introduciendo en un código de Python el título de nuestra obra y el número de poemas deseados nos cree un PDF con aquello que le hemos indicado.

Para ello se hizo uso de la librería de Python, python-docx [57], que nos da acceso a documentos words desde Python.

El programa se programa de forma que inicialmente se pide por pantalla al usuario el título de la obra y el número de poemas. Una vez se tiene el título de la obra se abre un nuevo documento Word con el nombre que se ha recibido. Una vez abierto el documento Word primero escribimos el título en la primera cara y luego mediante un bucle que se realizara tantas veces como poemas haya indicado el usuario se ira llamando al modelo y creando los poemas con la función generate_text. Se le da un título con su número a cada poema hasta que se tengan todos los poemas deseados.

Una vez escrita toda la obra mediante el comando de la librería doxc2pdf [58], convert, convertimos el Word a PDF y guardamos nuestra obra lista para enviar al editor que se desee.

Todo el código desarrollado se puede ver en ANEXO VIII.

Capítulo 6. ANÁLISIS DE RESULTADOS

Inicialmente cuando se realizó un primer modelo con secuencias en las que solo se pedía al modelo que aprendiera a obtener la siguiente palabra dada la secuencia se obtuvieron resultados peores a lo que se esperaba. Algunos de esos versos obtenidos por este modelo se observan en la figura Ilustración 34.

```
['flor', 'resplandor', 'quien', 'viento', 'nubes', 'playas']
['las', 'cielo', 'te', 'tierra', 'como', 'Es', 'de', 'boca', 'Se', 'hay', 'allí', ',', 'y']
['amor', 'o', 'los', 'bien', 'sólo', 'el', 'cielo']
['yo', 'hueso', 'luces', 'manos']
['puede', 'instante', '?']
```

Ilustración 34: resultados primer modelo [elaboración propia]

Al observar estos resultados podemos ver que tienen la longitud aproximada de los versos utilizados para el entrenamiento, y se aprecia el vocabulario poético del dataset, pero realmente no encontramos estructura gramatical o es muy pobre.

Sin embargo, al cambiar esto como se explica en el capítulo Secuencias en el modelo, la mejora fue muy notable y se empezaron a obtener resultados bastante buenos con mucha mayor estructura gramatical y mucho más parecido a algo que podríamos llamar poesía. Algunos de estos versos se observan en Ilustración 35.

```
rosa , robusta gastada o presentimiento ,
casi sagrada
sintiendo bóveda como La fría ,
como mirándome oigo .
```

Ilustración 35: resultados versos modelo mejorado [elaboración propia]

Se considero que los resultados obtenidos ya alcanzaban el objetivo propuesto en el proyecto. Este resultado puede y debe mejorarse optimizando la parametrización de la red neuronal, como se realizó en el proyecto de fin de grado paralelo a este por Venancio García Muñoz.

Finalmente, el modelo entrenado con secuencias de estrofas completas también elaboró estrofas que se consideró que eran aceptables y de la calidad que se esperaba alcanzar, además de que muchas veces tienen un tema común que se repite a lo largo de la estrofa. Una estrofa ejemplo es la de Ilustración 36.

```
Manos ilustre humana como el amor ,
que nunca se ve pasar ,
el vacío verde o verde o lágrima
que a veces aquí se yergue a un beso en un agua .
```

Ilustración 36: resultados estrofas modelo mejorado [elaboración propia]

Por lo que una vez demostrado que se puede realizar algo parecido a poesía con nuestro modelo, a pesar de que todavía debería pulirse mal e intentar minimizar los errores gramaticales, hemos de comprobar el resultado de su implementación en las aplicaciones planteadas.

Para ello activamos el bot programado y entramos a Twitter para ver si efectivamente está escribiendo los tweets.

Ilustración 37:Captura tweets Bot desde Twitter [elaboración propia]

Como observamos en la Ilustración 37, las estrofas creadas por nuestro modelo aparecen en Twitter con la frecuencia de 5 minutos establecida en este caso. Por lo que queda

demostrando que el modelo puede aplicarse de forma sencilla como elemento de un programa mayor que en este caso está programado para tuitear.

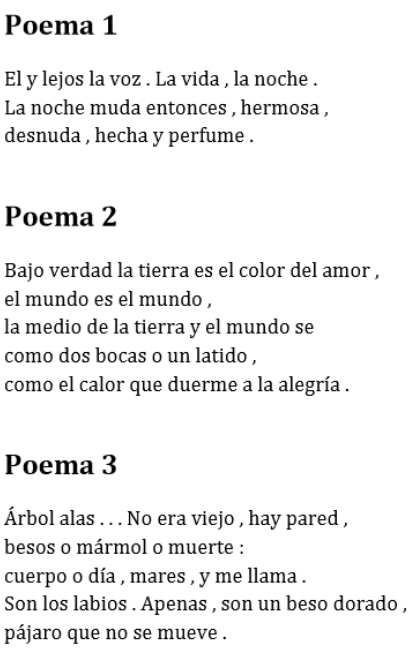
La otra forma en la que se implementó fue para crear un PDF con un numero de poemas seleccionados, por lo que se da run a este programa y se observan los resultados.

Inicialmente el programa nos pide por pantalla que introduzcamos el título de la obra y el número de poemas, como se observa en Ilustración 38.

```
introduzca el titulo de la obra: Obra Analisis de resultados
introduzca el numero de poemas: 5
```

Ilustración 38: Pantalla Python título y numero de poemas obra [elaboración propia]

Esperamos a que el programa acabe y comprobamos que efectivamente se nos ha guardado un PDF en la carpeta con nombre el título de la obra, con una portada con dicho título y el número de poemas seleccionado. En se muestran alguno de los poemas del PDF creado.



Poema 1

El y lejos la voz . La vida , la noche .
La noche muda entonces , hermosa ,
desnuda , hecha y perfume .

Poema 2

Bajo verdad la tierra es el color del amor ,
el mundo es el mundo ,
la medio de la tierra y el mundo se
como dos bocas o un latido ,
como el calor que duerme a la alegría .

Poema 3

Árbol alas . . . No era viejo , hay pared ,
besos o mármol o muerte :
cuerpo o día , mares , y me llama .
Son los labios . Apenas , son un beso dorado ,
pájaro que no se mueve .

Ilustración 39: Poemas creados en PDF [elaboración propia]

Se comprueba entonces que esta implementación también se ha realizado correctamente.

Por todo esto podemos afirmar que puede realizarse algo parecido a poesía con una red neuronal LSTM. Con el dataset y el entrenamiento necesario, se puede obtener algo que que parece entender la gramática y semántica española. Sigue habiendo errores de coordinación y la red neuronal no parece entender del todo algunos conceptos sintácticos. En cuanto al nivel poesía también podría mejorar, aunque a veces nos encontramos con imágenes y frases interesantes, generalmente las frases no tienen el nivel de la poesía de Vicente Aleixandre y nos encontramos con palabras repetidas y estrofas sin un tema demasiado coherente.

Capítulo 7. CONCLUSIONES Y TRABAJOS FUTUROS

La red LSTM es una herramienta potente de computación que permite a las redes neuronales afrontar el problema de lenguaje. De forma que con un dateset tratado de forma correcta, sin necesidad de que sea excesivamente amplio, pues solo hizo falta la obra de un autor, podemos entrenar la red neuronal y observar unos resultados decentes. Se consiguió obtener estrofas, formadas con versos con cierto sentido semántico y gramatical.

Sin embargo, hay margen de mejora. Sigue habiendo errores de coordinación y la red neuronal no parece entender del todo algunos conceptos sintácticos. En cuanto al nivel poesía también podría mejorar, aunque a veces nos encontramos con imágenes y frases interesantes, generalmente las frases no tienen el nivel de la poesía de Vicente Aleixandre y nos encontramos con palabras repetidas y estrofas sin un tema demasiado coherente.

Para solucionar eso la primera opción es seguir parametrizando la red, buscar que profundidad es la más conveniente, si es mejor tener una red bidireccional etc, trabajo en el que se centró Venancio García en su proyecto paralelo.

Para seguir impulsándolo también se podría aumentar el dataset, incluyendo a otros autores para tener más entrenamiento y que la red aprenda esos conceptos que sigue sin dominar. Con esto perderíamos el estilo característico como beneficio de usar otro autor, pero aumentaríamos el vocabulario y tendríamos un dataset mayor para el entrenamiento de conceptos sintácticos y gramaticales en la red.

También se podría llevar a otro nivel la red neuronal, implementando una GAN, con un discriminador que intente distinguir entre versos reales y versos creados por nuestro modelo. Ajustando nuestro modelo para que cada vez engañe mejor al discriminador, y mejorando el discriminador para que también sea cada vez más eficaz. Podríamos de esta forma en teoría mejorar nuestro modelo mediante este enfoque competitivo, como ya se ha demostrado en

inteligencias artificiales centrada en imágenes. El esquema de del concepto de GAN se muestra en Ilustración 40: Esquema GAN

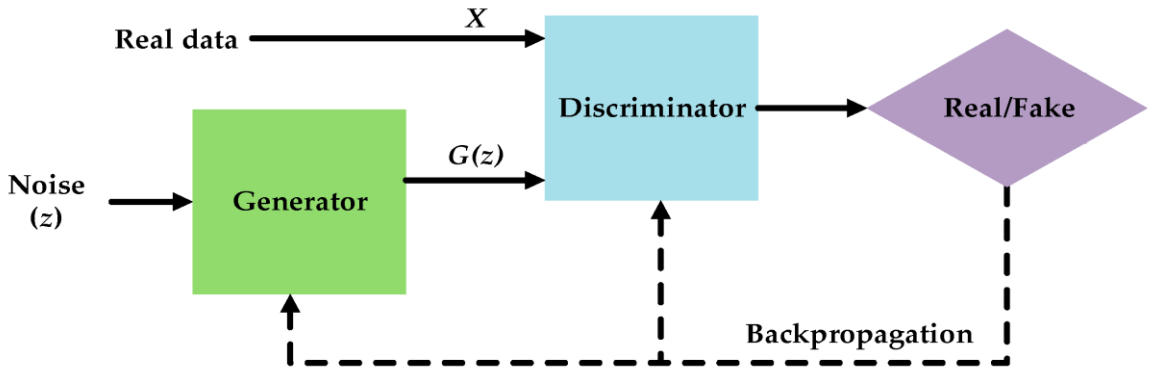


Ilustración 40: Esquema GAN

Por último, también podríamos meternos a más bajo nivel dentro de la LSTM, ajustando su configuración e incluso desarrollando nuevas versiones de redes neuronales para enfocar mejor nuestro problema.

Capítulo 8. BIBLIOGRAFÍA

[3] Satandfor University School of Business. "Andrew Ng: Artificial Intelligence is the new electricity". Youtube.com. Julio, 2020. https://www.youtube.com/watch?v=21EiKfQYZXc.

[4] Teresa Guerrero y Cristina G. Lucio. "Un nuevo aliado de los médicos". El mundo, Julio 2020. https://lab.elmundo.es/inteligencia-artificial/salud.html.

[5] Diego Calvo, "definición de una red neuronal artificial". diegocalvo.es, Julio 2020. https://www.diegocalvo.es/definicion-de-red-neuronal/#:~:text=autom%C3%A1tico%20%7C%201%20Comentario-,Definici%C3%B3n%20de%20red%20neuronal%20artificial,interconectadas%20entre%20s%C3%AD%20mediante%20enlaces.

[6] Thomas Wood, deepai.org. Julio, 2020, https://deepai.org/machine-learning-glossary-and-terms/softmax-layer.

[7] Deep Ai, deepai.org. Julio, 2020, https://deepai.org/machine-learning-glossary-and-terms/sigmoid-function#:~:text=A%20sigmoid%20function%20is%20a,in%20the%20prediction%20of%20probabilities.

[8] Science Direct, sciencedirect.com. Julio, 2020, https://www.sciencedirect.com/topics/computer-science/sigmoid-function

[9] CODE-AI, code-ai.mk, Julio 2020, https://code-ai.mk/neural-network-with-c-from-scratch/

[10] Aditi Mittall, towardsdatascience.com, Julio 2020, https://towardsdatascience.com/understanding-rnn-and-lstm-f7cdf6dfc14e.

[11] Chi Feng-Wang, towardsdatascience.com, Julio 2020, https://towardsdatascience.com/the-vanishing-gradient-problem-69bf08b15484.

[12] Nir Arbel, medium.com, Julio 2020, https://medium.com/datadriveninvestor/how-do-lstm-networks-solve-the-problem-of-vanishing-gradients-a6784971a577

[13] Stephen O'neal cointelegraph.com, Julio 2020, https://cointelegraph.com/explained/tokenization-explained

[14] Athif Shaffy, medium.com, Julio, 2020, https://medium.com/@athif.shaffy/one-hot-encoding-of-text-b69124bef0a7.

[15] Tensorflow, Julio, 2020 https://www.tensorflow.org/?hl=es-419

[16] Keras, Julio 2020, https://keras.io/

[17] David Foster, editorial O'reilly, 2019. "Generative Deep learning, teaching machines to write, compose and paint"

[18] Will Koehrsen, towardsdatascience.com, https://towardsdatascience.com/neural-network-embeddings-explained-4d028e6f0526#:~:text=Notebook%20on%20GitHub).-,Embeddings,vector%20representations%20of%20discrete%20variables.&text=As%20input%20to%20a%20machine%20learning%20model%20for%20a%20supervised%20task.

[19] Google colab, Julio 2020, https://colab.research.google.com/notebooks/mlcc/intro_to_sparse_data_and_embeddings.ipynb?utm_source=mlcc&utm_campaign=colab-external&utm_medium=referral&utm_content=embeddings-colab&hl=en

[20] Keras, Julio 2020, https://keras.io/api/losses/.

[21] Keras, Julio 2020, https://keras.io/api/losses/probabilistic_losses/#categoricalcrossentropy-class

[22] Peltarion, peltario.com, Julio 2020, https://peltarion.com/knowledge-center/documentation/modeling-view/build-an-ai-model/loss-functions/categorical-crossentropy

[23] Keras, Julio 2020, https://keras.io/api/optimizers/rmsprop/

[24] Rohith Gandhi, towardsdatascience.com, Julio 2020, https://towardsdatascience.com/a-look-at-gradient-descent-and-rmsprop-optimizers-f77d483ef08b

[25] Ayoosh Kathuria, paperspace.com, Julio 2020, https://blog.paperspace.com/intro-to-optimization-momentum-rmsprop-adam/

[26] Vitaly Bushaev, towardsdatascience.com, Julio 2020, https://towardsdatascience.com/understanding-rmsprop-faster-neural-network-learning-62e116fcf29a#:~:text=The%20central%20idea%20of%20RMSprop,moving%20average%20of%20squared%20gradients.

[27] Renu Kandelwal, medium.com, Julio 2020, https://medium.com/datadriveninvestor/overview-of-different-optimizers-for-neural-networks-e0ed119440c3#:~:text=How%20Gradient%20Descent%20helps%20achieve,%2C%20RMSProp%2C%20Adam%20and%20Nadam.

[28] Matthew Strewart, towardsdatascience.com https://towardsdatascience.com/neural-network-optimization-7ca72d4db3e0

[29] Jason Brownlee, Machine Learning Mastery, Julio 2020, https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch/#:~:text=The%20batch%20size%20is%20a%20number%20of%20samples%20processed%20before,samples%20in%20the%20training%20dataset.

[30] Twitter Developper, Twitter, Julio 2020, https://developer.twitter.com/en/docs/basics/authentication/oauth-1-0a/obtaining-user-access-tokens

[31] Twitter Developper, Julio 2020, https://developer.twitter.com/en/docs/basics/authentication/api-reference/authenticate

[32] Tweepy, tweepy.org, Julio 2020, https://www.tweepy.org/

[33] Twitter Developper, Twitter, Julio 2020, https://developer.twitter.com/en/docs/api-reference-index

[34] OpenAI, Julio 2020, https://openai.com/.

[35] WuTheFwasThat, github.com, Julio 2020, https://github.com/openai/gpt-2.

[36] OpenAI, openai.com, Julio 2020, https://openai.com/blog/openai-api/.

[37] Vox, "Computers just got a lot better at writing", Youtube.com, Julio 2020, https://www.youtube.com/watch?v=gcHkxP9adiM

[38] Allen Institute of technology, Julio 2020. https://demo.allennlp.org/next-token-lm?text=AllenNLP%20is

[39] RoBERTa, huggingface.co, Julio 2020, https://huggingface.co/openai-detector/.

[40] Allen Institute of technology, Julio 2020, https://grover.allenai.org/

[41] AI21labs, HAIM, Julio 2020, https://www.ai21.com/haim.

[42] AIdungeon, Julio 2020, https://play.aidungeon.io/

[43] Google, Poem portraits, Julio 2020, https://artsexperiments.withgoogle.com/poemportraits

[44] Daniel Castro, datainovation.org, Who is wining the Ai race?, https://www.datainnovation.org/2019/08/who-is-winning-the-ai-race-china-the-eu-or-the-united-states/

[45] Gabriel Caraballo, planetchatbot.com, Julio 2020, https://planetachatbot.com/lo-que-aprend%C3%AD-entrenando-al-primer-chatbot-gpt-2-en-espa%C3%B1ol-9644c236bf9a

[46] NIL, WASP Julio 2020, http://nil.fdi.ucm.es/?q=node/206

[47] Pablo Gervás, NIL, Julio 2020, http://nil.fdi.ucm.es/?q=members/pablogervas

[48] Pablo Gervás, google scholar, Julio 2020, https://scholar.google.com/citations?user=AcY-Y2gAAAAJ&hl=es

[49] Eduardo Geûens, Twitter, Julio 2020, https://twitter.com/BotPoeta.

[50] El mundo, Julio 2020, https://www.elmundo.es/espana/2019/12/18/5dfa081afc6c834c168b4572.html

[51] El club de los poetas muertos, Peter Weir, 1989.

[52] Coursera, Julio 2020, https://es.coursera.org/

[53] Vicente Alixandre, Poesi.as, Julio 2020https://www.poesi.as/Vicente_Aleixandre.htm

[54] Keras, LSTM, Julio 2020, https://keras.io/api/layers/recurrent_layers/lstm/

[55] Medium.com, Julio 2020, https://medium.com/@amarbudhiraja/https-medium-com-amarbudhiraja-learning-less-to-learn-better-dropout-in-deep-machine-learning-74334da4bfc5#:~:text=The%20term%20%E2%80%9Cdropout%E2%80%9D%20refers%20to,which%20is%20chosen%20at%20random.

[56] Keras, model class, Julio 2020, https://keras.io/api/models/model/

[57] Python-docx, documentation, Julio 2020, https://python-docx.readthedocs.io/en/latest/

[58] Pypi.org, docx2pdf, Julio 2020, https://pypi.org/project/docx2pdf/

[59] MDPI, Julio 2020, https://www.mdpi.com/2072-4292/12/7/1149/htm

[60] Integrateing the SDG´s into a TFG/TFM, José Luis Becerra & José Carlos Romero ICAI – School of engineering.

[61] Europapress, Julio 2020, https://www.europapress.es/epagro/noticia-cambio-climatico-europa-impulsara-inteligencia-artificial-contra-cambio-climatico-mediterraneo-punto-mira-20200721185841.html

ANEXO I: CREACION DICCIONARIO

```
1. import re
2.
3. def ampliaDiccionario(dict, palabra):        #Recibe un diccionario
   y una palabra
4.     vec=[];                                  #Creación del One Hot
   Vector
5.     a=0
6.
7.     for actual in dict:                      #Recorrido del
   diccionario buscando la palabra
8.         if ((actual.upper() == palabra.upper())and(a==0)):
9.             a=1
10.          if a==0 and len(palabra)>0:
11.              dict.append(palabra.lower())
12.              print("Nueva palabra "+ palabra)
13.          return vec                                #Devuelve One Hot
   Vector
14.
15.      def renderUnVerso(dict, verso):
16.
17.          x=re.split(r'(;|«|»|,|:|¿|\?|¡|!|>|<|-|_|—
   |\.|\(|\))\s*',verso)
18.
19.          pal_verso = [];
20.          pal_verso.append("sof")
21.
22.          for elemento in x:
23.              palabras = elemento.split(" ")
24.
25.              for palabra in palabras:
26.                  if (palabra != ''):
27.                      pal_verso.append(palabra)
28.
29.          pal_verso.append('eof')
30.          lista_sec=[]
31.          i=0
```

```
32.          for a in pal_verso:
33.              vec=ampliaDiccionario(dict, pal_verso[i])
34.              lista_sec.append(vec)
35.              i=i+1
36.
37.          return lista_sec
38.
39.      diccionario = []
40.      versos = open("EstrofasEOP.txt", 'r', encoding='windows-
   1252').read().split('\n')
41.
42.
43.      tot_secuencias=[]
44.      i=0
45.      for a in versos :
46.          secuencia=renderUnVerso(diccionario, versos[i])
47.          tot_secuencias.append(secuencia)
48.          i=i+1;
49.
50.      print(len(diccionario))
51.
52.      file=open('Diccionario_Estrofas.txt', 'w', encoding='windows-
   1252')
53.      i=0
54.      for indd in diccionario:
55.          file.write(diccionario[i])
56.          print("Nueva palabra "+ diccionario[i])
57.          file.write('\n')
58.
59.          i=i+1;
60.
61.
62.      print(len(versos));
63.
64.      file.close()
```

ANEXO II: TOKENIZACION VERSO

```
1. import csv
2. import re
3.
4. def createToken(dict, palabra):        #Recibe un diccionario y una
   palabra
5.
6.     i=0;
7.     a=0;
8.     token=0;
9.     for actual in dict:                    #Recorrido del
   diccionario buscando la palabra
10.             if actual.lower() == palabra.lower() and a==0:
11.                 token = i;                #Pone a 1 la posición de
   la palabra, resto a 0
12.                 a=1
13.                 print(actual)
14.             else:
15.                 i=i+1;
16.
17.         return token                              #Devuelve One
   Hot Vector
18.
19.     def renderUnVerso(dict, verso):
20.         x=re.split(r'(;|,|:|¿|\?|¡|!|>|<|-|_|—
   |\.|\(|\))\s*',verso)
21.
22.         pal_verso = [];
23.         pal_verso.append("sof")
24.
25.         for elemento in x:
26.             palabras = elemento.split(" ")
27.
28.             for palabra in palabras:
29.                 if (palabra != ''):
30.                     pal_verso.append(palabra)
31.
```

```
32.          pal_verso.append('eof')
33.
34.          lista_sec=[]
35.          i=0
36.          for a in pal_verso:
37.              vec=createToken(dict, pal_verso[i])
38.              lista_sec.append(vec)
39.              i=i+1
40.
41.          return lista_sec
42.
43.      diccionario = open('Diccionario_Nuevo.txt','r',encoding="wind
   ows-1252").read().split('\n')
44.      versos = open("obra.txt", 'r', encoding="windows-
   1252").read().split('\n')
45.
46.
47.      tot_secuencias=[]
48.      i=0
49.      for a in versos :
50.          secuencia=renderUnVerso(diccionario, versos[i])
51.          tot_secuencias.append(secuencia)
52.          i=i+1;
53.
54.      i=0
55.      myfile=open('Secuencias1.txt', 'w', newline='')
56.      for indd in tot_secuencias:
57.          wr = csv.writer(myfile, quoting=csv.QUOTE_ALL)
58.          wr.writerow(tot_secuencias[i])
59.          i=i+1;
60.
61.      myfile.close()
62.
63.      print(len(diccionario))
64.      print(len(tot_secuencias))
```

ANEXO III: TOKENIZACION ESTROFA

```
1. import csv
2. import re
3.
4. def createToken(dict, palabra):        #Recibe un diccionario y una
   palabra
5.
6.     i=0;
7.     a=0;
8.     token=0;
9.     for actual in dict:                    #Recorrido del
   diccionario buscando la palabra
10.             if actual.lower() == palabra.lower() and a==0:
11.                 token = i;                #Pone a 1 la posición de
   la palabra, resto a 0
12.                 a=1
13.                 #print(actual)
14.             else:
15.                 i=i+1;
16.
17.         return token                            #Devuelve One
   Hot Vector
18.
19.     def renderUnVerso(dict, verso):
20.         x = re.split(r'(;|—|,|:|¿|\?|¡|!|\.|\(|\))\s*', verso)
21.         pal_verso = [];
22.
23.         for elemento in x:
24.             palabras = elemento.split(" ")
25.
26.             for palabra in palabras:
27.                 if (palabra != ''):
28.                     pal_verso.append(palabra)
29.
30.         #pal_verso.append('eof')
```

```
31.          str1 = ""
32.          lista_sec=[]
33.          i=0
34.          tam_sec=0
35.          for a in pal_verso:
36.              vec=createToken(dict, pal_verso[i])
37.              lista_sec.append(vec)
38.              str1+=str(vec)
39.              str1+=" "
40.              i=i+1
41.              tam_sec +=1
42.
43.
44.          return str1,tam_sec
45.
46.      diccionario = open('Diccionario_Estrofas.txt','r',encoding="w
   indows-1252").read().split('\n')
47.      estrofas = open("EstrofasEOP.txt", 'r', encoding="windows-
   1252").read().split('eop')
48.
49.      sec_estrofas=[]
50.      tot_secuencias=[]
51.      isec=0
52.      for a in estrofas :
53.          versos =estrofas[isec].split('\n')
54.          i = 0
55.          estrofa=""
56.          tam_sec=0
57.          for a in versos :
58.              secuencia,tam_secuencia=renderUnVerso(diccionario, ve
   rsos[i])
59.              tot_secuencias.append(secuencia)
60.              estrofa += secuencia;
61.              i=i+1;
62.              tam_sec+=tam_secuencia
63.          estrofa += str(27)
64.          if tam_sec>300:
```

```
65.             print(tam_sec)
66.         li = list(estrofa.split(" "))
67.         sec_estrofas.append(li)
68.         isec+=1
69.
70.     i=0
71.     myfile=open('SecuenciasObraCompleta.txt', 'w', newline='')
72.     for indd in sec_estrofas:
73.         wr = csv.writer(myfile, quoting=csv.QUOTE_ALL)
74.         wr.writerow(sec_estrofas[i])
75.         i=i+1;
76.
77.     myfile.close()
78.
79.     print(len(diccionario))
80.     print(len(tot_secuencias))
```

ANEXO IV: CREACIÓN VECTORES DE ENTRENAMIENTO

```
1. token_list = open('SecuenciasObraCompleta.txt', 'r', encoding="windows-1252").read().replace('"', ' ').replace(',', ' ').split()
2. diccionario = open('Diccionario_Estrofas.txt', 'r', encoding="windows-1252").read().split('\n')
3.
4. for i in range(len(token_list)):
5.     token_list[i] = int(token_list[i])
6.
7. print("token")
8. print(token_list)
9. print(token_list[0])
10.
11.
12.     def generate_sequences(token_list, step, longitud,total_words):
13.         X = np.zeros((0,longitud),dtype=int)+27
14.         y = np.zeros((0,longitud),dtype=int)+27
15.         control_bucle = 0
16.         aux_fila = np.zeros((1,longitud),dtype=int)+27
17.         ctrl_fila = 0
18.         while(control_bucle<len(token_list)):
19.             if(token_list[control_bucle]!=27):
20.                 aux_fila[0,ctrl_fila]=token_list[control_bucle]
21.                 ctrl_fila += 1
22.             else:
23.                 ctrl_fila = 0
24.                 X = np.append(X,aux_fila,axis=0)
25.                 y = np.append(y,[np.append(aux_fila[0,1:], 27)], axis=0)
26.                 #padded_sequences = pad_sequences(sequences, maxlen=max_length, padding='post')
27.                 aux_fila[0,:]=27
28.             control_bucle+=1
29.         y_out = np.zeros((len(X),longitud,total_words),dtype=int)
```

```
30.          for i in range(len(X)):
31.              for j in range(longitud):
32.                  y_out[i][j][y[i][j]]=1
33.          #y = np_utils.to_categorical(y, len(diccionario))  #
   Converts a class vector (integers) to binary class matrix.
34.
35.          num_seq = len(X)
36.
37.          return X, y_out, num_seq
```

ANEXO V: CREACIÓN Y ENTRENAMIENTO RED NEURONAL

```
1. import numpy as np
2. import keras
3. from keras.utils import np_utils
4.
5.
6. token_list = open('SecuenciasObraCompleta.txt', 'r', encoding="wind
   ows-1252").read().replace('"', ' ').replace(',', ' ').split()
7. diccionario = open('Diccionario_Estrofas.txt', 'r', encoding="windo
   ws-1252").read().split('\n')
8.
9. for i in range(len(token_list)):
10.          token_list[i] = int(token_list[i])
11.
12.      print("token")
13.      print(token_list)
14.      print(token_list[0])
15.
16.
17.      def generate_sequences(token_list, step, longitud,total_words
   ):
18.          X = np.zeros((0,longitud),dtype=int)+27
19.          y = np.zeros((0,longitud),dtype=int)+27
20.          control_bucle = 0
21.          aux_fila = np.zeros((1,longitud),dtype=int)+27
22.          ctrl_fila = 0
23.          while(control_bucle<len(token_list)):
24.              if(token_list[control_bucle]!=27):
25.                  aux_fila[0,ctrl_fila]=token_list[control_bucle]
26.                  ctrl_fila += 1
27.              else:
28.                  ctrl_fila = 0
29.                  X = np.append(X,aux_fila,axis=0)
30.                  y = np.append(y,[np.append(aux_fila[0,1:], 27)],
   axis=0)
```

```
                    #padded_sequences = pad_sequences(sequences, maxlen=max_length, padding='post')
                    aux_fila[0,:]=27
                control_bucle+=1
            y_out = np.zeros((len(X),longitud,total_words),dtype=int)
            for i in range(len(X)):
                for j in range(longitud):
                    y_out[i][j][y[i][j]]=1
            #y = np_utils.to_categorical(y, len(diccionario))  # Converts a class vector (integers) to binary class matrix.

            num_seq = len(X)

            return X, y_out, num_seq


        step = 1;


        X, y, num_seq = generate_sequences(token_list, step,480,len(diccionario))


        ## ARCHITECTURE

        import tensorflow as tf
        from tensorflow.keras.layers import Dense, LSTM, Input, Embedding, Dropout, Bidirectional
        from tensorflow.keras.models import Model
        from tensorflow.keras.optimizers import RMSprop



        n_units = 256
        n_units2 = 128
        n_units3 = 32
        embedding_size =100
```

```
64.       text_in = Input(shape=(None,))
65.       #print(text_in)  # float32
66.
67.       # probando cosas
68.
69.       #print(text_in[1])  # float32
70.       #print(text_in[2])  # float32
71.
72.       total_words = len(diccionario)
73.       x = Embedding(total_words, embedding_size)(text_in)
74.       #x = keras.models.Sequential()
75.       #x.add(keras.layers.Embedding(total_words, embedding_size))
76.       #x.add(keras.layers.Bidirectional(keras.layers.LSTM(n_units))
   )
77.       x = LSTM(n_units, dropout=0.2, recurrent_dropout=0.2, return_
   sequences=True)(x)
78.       #x = LSTM(n_units, dropout=0.2, recurrent_dropout=0.2,
   return_sequences=True)(x)
79.       #x = LSTM(n_units, dropout=0.2, recurrent_dropout=0.2,
   return_sequences=True)(x)
80.       #x = Bidirectional(LSTM(n_units, dropout=0.2,
   recurrent_dropout=0.2, return_sequences=True))(x)
81.       #x = SeqSelfAttention(attention_activation='sigmoid',
   return_attention=True)(x)
82.       #x = Dropout(0.25)(x)
83.       #x = Bidirectional(LSTM(n_units2))(x)
84.       #x.add(SeqSelfAttention(attention_activation='sigmoid'))
85.       #print(x)  # Shape(None, 256) float32
86.
87.       #x = Dropout(0.25)(x)#randomly setting a fraction, helps
   prevent overfitting.
88.       #print(x)  # Shape(None,256)
89.
90.       #print("tipo datos")
91.       #print(x.dtype)
92.       #print(x[1])
93.       #print(x.shape)
94.       #print(x[256])
```

```
95.
96.      text_out = Dense(total_words, activation='softmax')(x)
97.      print("Total words", total_words)
98.      #x.add(Dense(total_words,activation='softmax'))
99.      # ERROR 1.Warning(Optimización no importante¿?) 2.op_kernel->problema formato inpput data
100.     # Dense implements the operation: output = activation(dot(input, kernel) + bias) where activation
101.     # is the element-wise activation function passed as the activation argument, kernel is a weights matrix created by the layer,
102.     # and bias is a bias vector created by the layer (only applicable if use_bias is True).
103.
104.
105.     model = Model(text_in, text_out)
106.     opti = RMSprop(lr=0.001)
107.     model.compile(loss='categorical_crossentropy', optimizer=opti)
108.     epochs = 200
109.     batch_size = 16
110.     model.summary()
111.     model.fit(X, y, epochs=epochs, batch_size=batch_size, shuffle=True)  # callback para que no dure tanto en entrenar el modelo
112.
113.     model.summary()
114.     model.save('Estrofas_1capa_n256_2.h5');
```

ANEXO VI: GENERACIÓN TEXTO

```
1. import numpy as np
2. from keras.utils import np_utils
3. from keras.preprocessing.sequence import pad_sequences
4. import tensorflow as tf
5. import time
6.
7. #model= tf.keras.models.load_model('alvaroDropout_02_Bidi.h5')
8.
9. model= tf.keras.models.load_model('modelo_bot.h5')
10.      #token list
11.
12.      token_list = open('Secuencias1.txt','r',encoding="windows-
   1255").read().replace('"',' ').replace(',',' ').split()
13.      diccionario = open('Diccionario_Nuevo.txt','r',encoding="wind
   ows-1252").read().split('\n')
14.
15.      for palabra in range(len(diccionario)):
16.          if(diccionario[palabra] == 'eof'):
17.              eof=palabra
18.          if(diccionario[palabra] == 'inicio'):
19.              inicio=palabra
20.
21.      for i in range(len(token_list)):
22.          token_list[i]= int(token_list[i])
23.      #funciones
24.
25.      def sample_with_temp(preds, temperature):
26.          # helper function to sample an index from a probability
   array
27.          preds = preds[-1,:]
28.          preds = np.asarray(preds).astype('float64')
29.          preds = np.log(preds) / temperature
30.          exp_preds = np.exp(preds)
31.          preds = exp_preds / np.sum(exp_preds)
32.          probs = np.random.multinomial(1, preds, 1)
33.          return np.argmax(probs)
34.
35.      #def sample_wo_temp(preds)
36.      #    return np.random.
37.
38.      def generate_text(seed_text, next_words, model, max_sequence_
   len, temp, decay):
39.
40.
41.          output_word=[seed_text]
42.          output_text=[seed_text]
43.
44.          while output_word != eof
   & len(output_text)<max_sequence_len:
45.          #for _ in range(10):
46.
```

```
47.                 token_list=[output_text]
48.
49.                 probs = model.predict(tf.convert_to_tensor(token_list
    ), verbose=0)[0]
50.                 #aux =
    model.predict(tf.convert_to_tensor(token_list), verbose=0)
51.                 y_class = sample_with_temp(probs, temperature = temp)
52.                 temp = np.maximum(0.5,temp * decay)
53.
54.                 output_word = y_class
55.                 if output_word == eof:#"eof"
56.                     break
57.
58.                 output_text.append(output_word)
59.                 probs2 = probs
60.
61.             return output_text
```

ANEXO VII: BOT TWITTER

```
1. import tweepy
2. import time
3. import generate_text_bucle_mejorado
4. import tensorflow as tf
5.
6.
7.
8. # Authenticate to Twitter
9. auth = tweepy.OAuthHandler("-------------------------",
10.           "----------------------------------------------")
11.      auth.set_access_token("-------------------------------------
   ----------------",
12.           "-----------------------------------------")
13.
14.      api = tweepy.API(auth, wait_on_rate_limit=True)#wait_on
   rate_limit(la api está limitada)
15.
16.      try:
17.          api.verify_credentials()
18.          print("Authentication OK")
19.      except:
20.          print("Error during authentication")
21.
22.      #inicializacion
23.
24.      #WHILE MAIN
25.      model=tf.keras.models.load_model('Estrofas_ObraCompleta.h5')
26.      diccionario = open('Diccionario_Estrofas.txt', 'r', encoding=
   "windows-1252").read().split('\n')
27.
28.      while(1):
29.
30.          print("iteracion:")
31.
```

```
        diccionario = open('Diccionario_Estrofas.txt', 'r', encoding="windows-1252").read().split('\n')

        for palabra in range(len(diccionario)):
            if (diccionario[palabra] == 'eop'):
                eop = palabra
            if (diccionario[palabra] == 'sop'):
                sop = palabra

        a=0

        while(a==0):
            generated=generate_text_bucle_mejorado.generate_text(sop,20,model,350,2.3,0.77)

            verso = ""
            palabra_anterior = 'palabra_anterior'
            for i in generated:
                if (diccionario[i] != 'sop' and diccionario[i] != 'inicio'):
                    if (diccionario[i] == 'eof'):
                        diccionario[i] = "\n"
                        verso += diccionario[i]
                        diccionario[i] = 'eof'
                    else:
                        if (palabra_anterior == "." or palabra_anterior == "palabra_anterior" or palabra_anterior == "?" or palabra_anterior == "!"):
                            # verso.append(diccionario[i])
                            palabra = diccionario[i].capitalize()
                            verso += palabra + " "
                        else:
                            verso += diccionario[i] + " "

                        palabra_anterior = diccionario[i]

            print(verso)
            print("\n")
```

```
65.
66.             #160 caracteres
67.             if(len(verso) <= 160 and len(verso)>2):
68.                 a=1
69.                 print("verso valido")
70.
71.         api.update_status(verso)
72.         print("verso escrito")
73.
74.         time.sleep(300)
75.
76.         del verso
77.         del generated
```

ANEXO VIII: CREADOR OBRAS

```
1. from docx import Document
2. import generate_text_bucle_mejorado
3. import tensorflow as tf
4. from docx2pdf import convert
5. from docx.enum.text import WD_ALIGN_PARAGRAPH
6. from docx.shared import Pt
7.
8. document = Document()
9.
10.      titulo = input("introduzca el titulo de la obra: ")
11.      titulo=titulo.capitalize()
12.      p=document.add_paragraph()
13.      p.paragraph_format.alignment = WD_ALIGN_PARAGRAPH.CENTER
14.      r=p.add_run(titulo)
15.      r.font.size = Pt(40)
16.      r.bold = True
17.
18.      p = document.add_paragraph('de Ixanel Avecendreti')
19.      p.alignment = 1
20.
21.      document.add_page_break()
22.
23.      poemas = input("introduzca el numero de poemas: ")
24.      poemas=int(poemas)
25.      print(poemas)
26.
27.      model=tf.keras.models.load_model('Estrofas_ObraCompleta.h5')
28.      diccionario = open('Diccionario_Estrofas.txt', 'r', encoding=
   "windows-1252").read().split('\n')
29.
30.      for palabra in range(len(diccionario)):
31.          if (diccionario[palabra] == 'eop'):
32.              eop = palabra
33.          if (diccionario[palabra] == 'sop'):
```

```
34.                 sop = palabra
35.
36.         h=0;
37.
38.         while(h<poemas):
39.
40.             h=h+1;
41.             print("iteracion:")
42.
43.             a=0;
44.             num_eof=0;
45.
46.             while(a==0):
47.                 generated=generate_text_bucle_mejorado.generate_text(
    sop,20,model,350,2.3,0.84)
48.
49.                 verso = ""
50.                 palabra_anterior = 'palabra_anterior'
51.                 for i in generated:
52.                     if (diccionario[i] != 'sop' and diccionario[i] !=
     'inicio'):
53.                         if (diccionario[i] == 'eof'):
54.                             diccionario[i] = "\n"
55.                             num_eof=num_eof+1
56.                             verso += diccionario[i]
57.                             diccionario[i] = 'eof'
58.                         else:
59.                             if (palabra_anterior == "." or palabra_an
    terior == "palabra_anterior" or palabra_anterior == "?" or palabra_
    anterior == "!"):
60.                                 # verso.append(diccionario[i])
61.                                 palabra = diccionario[i].capitalize()
62.                                 verso += palabra + " "
63.                             else:
64.                                 verso += diccionario[i] + " "
65.
66.                             palabra_anterior = diccionario[i]
```

```
67.
68.             print(verso)
69.             print("\n")
70.
71.             punto=(len(verso)-3)
72.
73.             #3 control calidad
74.             if(len(verso)>4):
75.                 if(num_eof>3 and (verso[punto]=='.' or verso[punto]=='!' or verso[punto]=='?')):
76.                     a=1
77.                     print("verso valido")
78.
79.
80.         titulo_poema= 'Poema '+ str(h)
81.         t=document.add_paragraph()
82.         p = t.add_run(titulo_poema)
83.         p.font.size = Pt(15)
84.         p.bold=True
85.         document.add_paragraph(verso)
86.
87.
88.         del verso
89.         del generated
90.
91.     titulo= titulo + '.docx'
92.     document.save(titulo)
93.     convert(titulo)
```

ANEXO IX: OBJETIVOS DESARROLLO SOSTENIBLE

Los objetivos del desarrollo sostenible (ODS) son una colección de 17 objetivos globales diseñados para ser un "guía para lograr un futuro mejor y más sostenible para todos ".

Existe una variedad de objetivos planteados de desarrollo sostenible, todos de extrema importancia y que deben ser tratados y prioritarios si queremos construir una sociedad mejor. Entre estos objetivos esta la reducción de la pobreza y la exclusión social, conseguir una educación primaria universal, promover la igualdad de género, reducir la mortalidad infantil, mejorar la sanidad maternal, combatir el sida y la tuberculosis, asegurarnos de crear un medioambiente sostenible y la asociación para el desarrollo.

Aunque el proyecto no se centra en la creación de nuevas tecnologías sostenibles o está centrado directamente en uno de los objetivos de desarrollo sostenible, si se quiere destacar la importancia de estos objetivos y como incluso en un proyecto de estas características ha de tenerse en cuenta. Sea cual sea el proyecto que realizamos siempre deben tenerse en cuenta los objetivos de desarrollo sostenible por su importancia y categoría.

Para empezar, uno de los objetivos de este proyecto es el impulsar las inteligencias artificiales y descubrir sus límites. Se cree firmemente que las inteligencias artificiales pueden ayudarnos al estudio del medioambiente y la creación de entornos más sostenibles. Ya se han empezado a realizar algunos proyectos que demuestran esto, como es “DeepCube”, una inteligencia artificial diseñada para combatir el cambio climático en el mediterráneo [61] y pretende contribuir a mejorar la comprensión de los procesos que sufre la Tierra en el contexto actual de cambio climático. Por lo que desarrollando nuestra inteligencia artificial esperamos impulsar la IA en general para que en el futuro se realicen muchos más proyectos de este tipo.

Por otro lado, el mayor gasto de energía del proyecto será la computación de los modelos, donde necesitaremos, a veces durante horas, mantener encendida nuestra herramienta de

computación consumiendo energía. Esto ha de tenerse en cuenta a lo largo del proyecto, y ser consciente de ello cuando se computa un modelo. Por lo que debe computarse aquello que sabemos de lo que se obtendrán buenos resultados o que nos ayudara a mejorar el proyecto. También debe tenerse en cuenta que al tratarse de poesía que modelo es mejor o peor puede ser subjetivo, esto nos permite cierta flexibilidad en cuanto a la elección de que modelo es mejor, y por lo tanto antes dos modelos parecidos, se dará prioridad a aquel que requiera menos computación y por lo tanto este más dirigido al ámbito del desarrollo sostenible.

Por último, este proyecto también trata la poesía, de la que decía Antonio Machado "la poesía es el diálogo del hombre, de un hombre con su tiempo", por lo que se invita a realizar ese dialogo interno con nosotros mismos y los problemas globales que nos rodean y debemos hacer frente, dándonos cuenta de la importancia y la necesidad de abordarlos y hacerlos frente para construir un futuro mejor.