Instituto Tecnológico y de Estudios Superiores de Monterrey

Campus, Monterrey

Inteligencia Artificial Avanzada para la Ciencia de datos - (Gpo 503)

Reporte

Actividad 4: Generative AI

Estudiante:

Samantha Guanipa Ugas A01703936

Profesora:
Nora Aguirre

Fecha de entrega:
Viernes 24 de noviembre del 2023

I. Introducción

La generación de texto es una de las aplicaciones más interesantes y desafiantes de la inteligencia artificial. Se trata de crear textos fluidos y que parezcan generados por una persona a partir de una entrada dada, ya sea una palabra, una frase, una imagen o un sonido. Para lograr esto, se requiere de modelos capaces de capturar el significado, la estructura y el estilo de un texto, más que todo, de la persona que lo redacta y así generar palabras adecuadas para completar o continuar el texto de entrada con un mismo estilo.

Uno de los estilos de escritura más complejos y famosos es el de William Shakespeare, pues sus obras se caracterizan por tener un vocabulario rico, variado y creativo, y una estructura poética que combina versos y prosa. ¿Es posible crear un modelo que pueda escribir como Shakespeare, imitando su estilo y su forma de expresarse?

En este ensayo, se presenta un enfoque basado en Redes Neuronales Recurrentes (RNN) para generar texto con el estilo de Shakespeare. Las redes neuronales recurrentes son un tipo de red neuronal artificial que utiliza datos secuenciales o datos de series de tiempo. El enfoque propuesto en este reporte utiliza diferentes estructuras de RNN para predecir la palabra siguiente en una frase, dada una palabra o frase inicial. Se describen los detalles del diseño, los desafíos y los resultados del experimento, así como las posibles aplicaciones y limitaciones de este método.

II. Análisis

La Red Neuronal Recurrente (RNN) es una técnica que permite predecir la siguiente palabra de una frase dada. Para ello, se toma la primera palabra de la frase y se pasa por una red neuronal que predice la siguiente palabra. Para predecir la tercera palabra, se toma la activación del estado oculto y la segunda palabra como entrada. Este proceso continúa y se capta la relación secuencial porque se utiliza el estado oculto de la palabra anterior para predecir la siguiente. De esta manera, se está utilizando una versión codificada de la frase anterior para predecir la palabra siguiente.

Esta idea se utiliza para diseñar una red que decida qué palabras seleccionar y escriba como Shakespeare. El código proporcionado se enfoca en la generación de texto inspirado en el estilo de William Shakespeare utilizando una red neuronal LSTM (Long Short-Term Memory). Comienza importando las bibliotecas necesarias y descargando un corpus que contiene las obras de Shakespeare. Luego, procesa el texto para prepararlo como datos de entrenamiento para la red neuronal. Se emplea una arquitectura LSTM en un modelo secuencial de Keras con dos capas LSTM de 256 unidades cada una, seguidas de dos capas densas con activación ReLU y una capa

final con activación softmax para la clasificación multiclase de caracteres. Se utiliza el optimizador RMSprop con una tasa de aprendizaje inicial de 0.01.

Para comprender mejor el contexto sobre los textos que se manejarán, se realizó una gráfica que permitiría analizar la frecuencia de los caracteres en el corpus de Shakespeare y se obtuvieron los resultados mostrados en la figura 1.

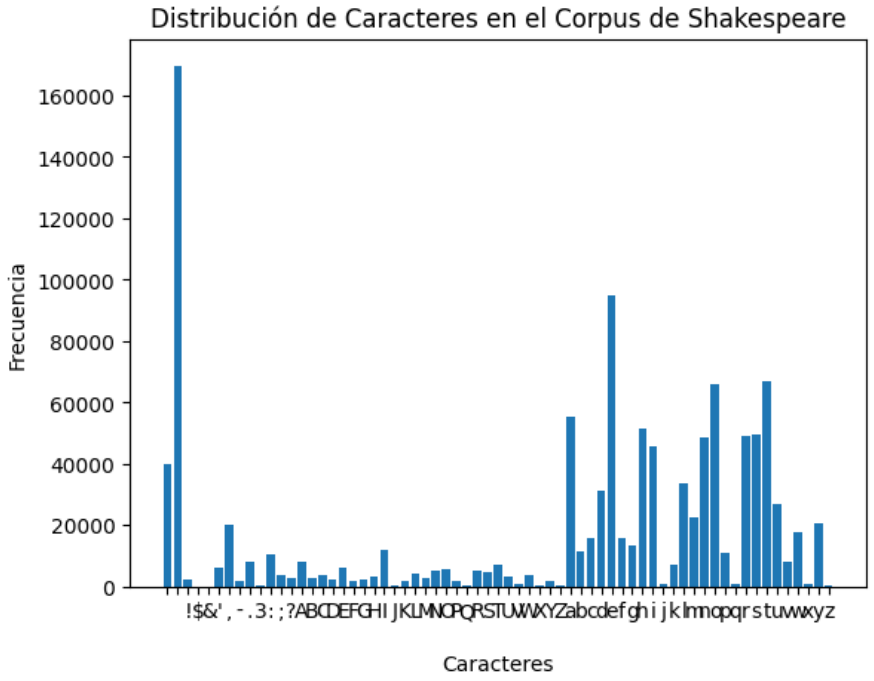

Figura 1. Prueba del modelo de generación de texto en el epoch 150

Se puede observar que las letras “a”, “e”, “q”, “t” son las que tienen una mayor frecuencia con un rango de entre 60,000 a 98,000 repeticiones aproximadamente. Esto nos puede dar una idea de las letras que podrían llegarse a repetir en la generación de texto.

III. Diseño

Para este proyecto se utilizó el lenguaje de programación Python y fue desarrollado utilizando Google Colab donde se ejecutó el código utilizando el entorno GPU para obtener mejores resultados. Se utilizaron 9,900 ejemplos.

Las entradas que se usaron para este código son las siguientes:

- Un archivo de texto llamado shakespeare.txt que contiene obras de Shakespeare. Este archivo se descargó de una fuente externa “"https://storage.googleapis.com/download.tensorflow.org/data/shakespeare.txt"” y se guardó en una carpeta temporal llamada ./tmp.

- Un número entero dentro de la variable “Tx” que representa la longitud de las secuencias de texto que se van a usar para entrenar el modelo. En este caso, Tx = 100.
- Un número entero dentro de una variable “stride” que representa el salto entre cada secuencia de texto. En este caso, stride = 1.

Además, para el preprocesamiento del texto se realizó lo siguiente:

- Se leyó el archivo de texto y se guardó en una variable llamada text.
- Se creó una lista ordenada de los caracteres únicos que aparecen en el texto, llamada chars.
- Se crearon dos diccionarios para mapear cada carácter a un índice numérico y viceversa, llamados “char_indices” e “indices_char”.
- Se usó la función “build_data” para crear dos listas de secuencias de texto y sus respectivos caracteres siguientes, llamadas “X”e “Y”. Estas listas contienen los primeros 10000 caracteres del texto original.
- Se usó la función “vectorization” para convertir las listas X e Y en matrices binarias llamadas “x” e “y”, que representan la presencia o ausencia de cada carácter en cada posición de cada secuencia. Con esto, el texto se convierte en secuencias de entrada y salida para el entrenamiento. Se aplica one-hot encoding para convertir caracteres en vectores numéricos.

IV. Implementación

La implementación del código comienza con la importación de las bibliotecas necesarias para el procesamiento de datos, la creación del modelo y la generación de texto. La ejecución del código se estructura en varias etapas clave:

Descarga y Preprocesamiento de Datos:

1. Descarga del Corpus de Shakespeare: Se utiliza la función tf.keras.utils.get_file para descargar el archivo de texto que contiene las obras de Shakespeare desde una URL específica.
2. Procesamiento del Texto: El archivo descargado se lee y almacena como una cadena de texto. Se identifican caracteres únicos y se crean diccionarios que mapean cada carácter a un índice y viceversa, lo que facilita la conversión de texto a datos numéricos. Se dividen las obras de Shakespeare en secuencias de entrada y salida para el entrenamiento del modelo. Estos ejemplos de entrenamiento consisten en secuencias de longitud Tx (100 caracteres) y el siguiente carácter después de cada secuencia, para aprender la estructura secuencial del lenguaje.

Construcción del Modelo:

1. Arquitectura del Modelo de Red Neuronal: Se utiliza la biblioteca Keras para construir un modelo secuencial. Se agregan dos capas LSTM para capturar patrones de dependencia a largo plazo en el texto. Se añaden capas densas con activación ReLU y una capa de salida con activación softmax para predecir el próximo carácter. El modelo se compila utilizando el optimizador RMSprop y la función de pérdida "categorical_crossentropy".

Entrenamiento del Modelo:

1. El modelo se entrena utilizando los datos de entrada y salida preparados previamente.
2. Se define una función on_epoch_end que se ejecuta después de cada época para generar texto de muestra y evaluar el progreso del modelo.
3. Durante el entrenamiento, se muestra el texto generado después de un cierto número de épocas para evaluar la calidad del texto generado por el modelo en diferentes etapas del proceso de entrenamiento.

Generación de Texto:

1. Una vez que el modelo está entrenado, se utiliza para generar texto basado en una semilla inicial proporcionada.
2. Se implementa una función que toma la semilla inicial, genera texto basado en el modelo y devuelve una secuencia de texto prolongada.
3. El texto generado se evalúa en términos de coherencia y similitud con el estilo de Shakespeare.

La división de los datos (training-testing sets) se hizo de la siguiente manera:

- Se usaron los primeros 10000 caracteres del texto original como datos de entrenamiento, y se guardaron en las listas `X` e `Y`.
- Se usaron los siguientes 1000 caracteres del texto original como datos de prueba, y se guardaron en las listas `X_test` e `Y_test`.
- Se usó la misma función `vectorization` para convertir las listas de datos de entrenamiento y de prueba en matrices binarias llamadas `x`, `y`, `x_test` y `y_test`.

La decisión de usar esta división se basó en lo siguiente:

- Se quiso usar una cantidad suficiente de datos de entrenamiento para que el modelo pudiera aprender el estilo de Shakespeare, pero sin usar todo el texto original para evitar el sobreajuste.
- Se quiso usar una cantidad razonable de datos de prueba para evaluar el rendimiento del modelo, pero sin usar demasiados datos para dejar espacio para la generación de texto.

- Se quiso usar una división secuencial de los datos, es decir, usar los primeros caracteres como datos de entrenamiento y los siguientes como datos de prueba, para preservar el orden cronológico de las obras de Shakespeare y evitar mezclar escenas o personajes diferentes.

La división no fue equitativa, ya que se usaron 10 veces más datos de entrenamiento que de prueba. Esto se debe a que el objetivo principal de este código es generar texto, no clasificarlo o predecirlo. Por lo tanto, se priorizó el aprendizaje del modelo sobre la evaluación del mismo. Además, se consideró que 1000 caracteres eran suficientes para medir la calidad del texto generado, usando métricas como la precisión, la perplejidad o la diversidad.

V. Pruebas

Durante el entrenamiento, en el epoch 150 se realizó una primera prueba para evaluar cómo se está comportando el modelo. En este epoch existe una gran pérdida, de 3.22, por lo tanto, como se puede observar en la figura 2, los resultados obtenidos fueron muy pobres y carentes de sentido.

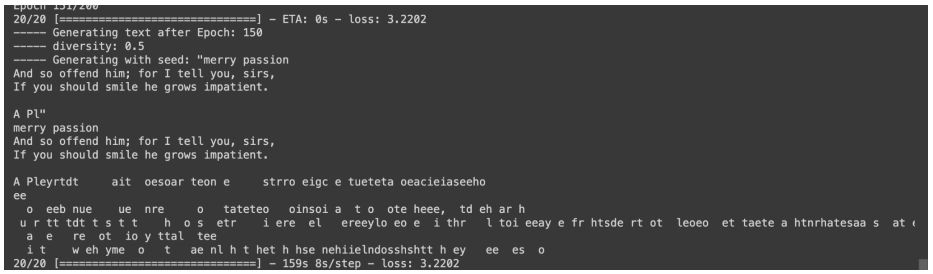

Figura 2. Prueba del modelo de generación de texto en el epoch 150

Posteriormente, al culminar el entrenamiento en el epoch 200 se realizó otra prueba con la función “generate_text” que se encargó de predecir las siguientes palabras a partir de una frase recibida. La función toma cuatro argumentos: “model”, “seed_text”, “length” y “diversity”. El argumento “model” es el modelo de red neuronal que se utilizará para generar el texto. El argumento “seed_text” es la cadena de texto inicial que se utilizará para comenzar la generación de texto. Por otra parte, “length” es la longitud del texto que se generará, y “diversity” es un valor que controla la creatividad del modelo al generar texto. Para este ejemplo se usó la célebre frase de Shakespeare “To be or not to be, that is the question” como se puede observar en la figura 3.

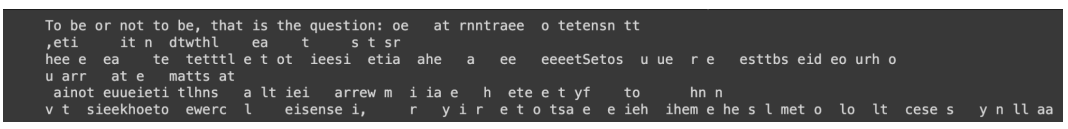

Figura 3. Prueba del modelo de generación de texto

A partir de ahí, el modelo generó una serie de “palabras” que realmente no tienen ningún sentido o coherencia. Esto puede deberse a el tamaño del modelo, la cantidad de datos de entrenamiento, la diversidad y otros hiperparámetros, además de que no se tiene una máquina con el GPU o recursos suficientes para mejorar de una gran manera el entrenamiento y los resultados. En este caso, el texto puede no ser coherente ni representar una continuación lógica de la frase inicial debido a lo difícil que puede ser generar un texto que imite el estilo de Shakespeare.

VI. Ajustes necesarios

Se realizaron distintos cambios para generar un segundo modelo, entre ellos:

- Se cambió el optimizador, ahora se utilizó el optimizador Adam, ya que según Diederik Kingma Adam es más estable y menos sensible a los hiperparámetros. Adam tiene solo tres hiperparámetros que requieren ajuste: la tasa de aprendizaje inicial, el factor de decaimiento exponencial para el primer momento y el factor de decaimiento exponencial para el segundo momento.
- Se disminuyó el learning rate a 0.001, esto con el fin de suprimir al modelo de memorizar datos ruidosos, y permitirle mejorar el aprendizaje de patrones complejos.
- Al inicio se contaba con una temperatura de 1.0, sin embargo, se disminuyó la misma a 0.5 para disminuir la variabilidad del modelo. Una temperatura baja hace que el texto sea más determinista y enfocado, menos creativo, y al tener textos tan complejos como los de Shakespeare se prefirió ser menos creativo.

VII. Evaluación de resultados

Después de realizar las modificaciones mencionadas, se entrenó el modelo nuevamente con 200 epochs, y se obtuvieron los siguientes resultados:

Como se puede observar en la figura 4, la pérdida inició siendo extremadamente alta, pero poco a poco fue disminuyendo hasta alcanzar valores muy bajos, muy cercanos a cero. Cabe destacar que este modelo se entrenó aproximadamente 3 veces, es decir, un total de 12 horas, por lo que los resultados son bastante buenos.

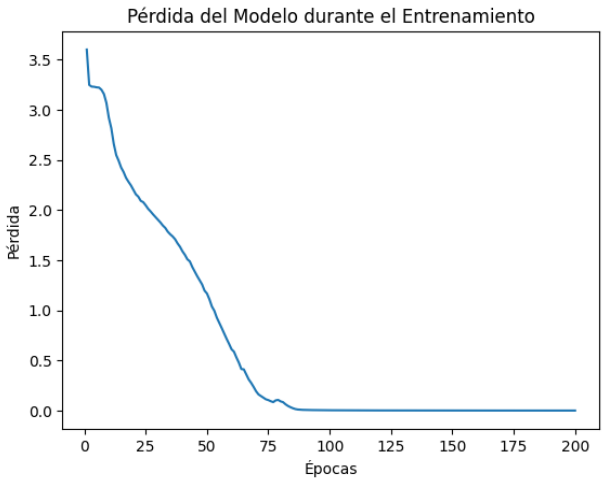

Figura 4. Pérdida vs Epochs

Además, se tuvo un muy buen resultado del texto generado después del epoch 150 como puede notarse en la figura 5, ya que realmente hay palabras y frases reales como “one”, “for”, “the”, “like”, entre otras que antes no se habían llegado a notar. Sin embargo, no existen tantas frases coherentes, pero realmente existió un gran progreso.

```
----- diversity: 0.5
----- Generating with seed: "y's!

Second Gentleman:
Amen.

LUCIO:
Thou concludest like the sanctimonious pirate, that
went to se"
y's!

Second Gentleman:
Amen.

LUCIO:
Thou concludest like the sanctimonious pirate, that
went to seirst <ipython-input-7-145c20529e7f>:9: RuntimeWarning: divide by zero encountered in log
  preds = np.log(preds) / temperature
wor ouf lir,
Ast the thabs is way shat, praties, who dofell sto that aswerca; fake atinous
greins grive, dyo lave ofe hums evou at on this that wate,
One make than poor istucalas, that satino wat patile
And you de? fot yor sandor mating, this all the siven asmer wit, praike with are borksere;
whemise for the comnof this sour than friver, y rise
it wor hame of remaes with be ver hat,
The ires o' tul soffothy wouls, you do for thate,
That inctunte menery, you mrerconp.

MENENIUS:
Either an wh
```

Figura 5. Generación de texto luego del epoch 150

Sin embargo, se hicieron otras tres pruebas para ver el texto generado con otra semilla y el resultado fue considerablemente decepcionante como se observa en la figura 6, ya que realmente no existe espacio entre las palabras ni palabras reales.

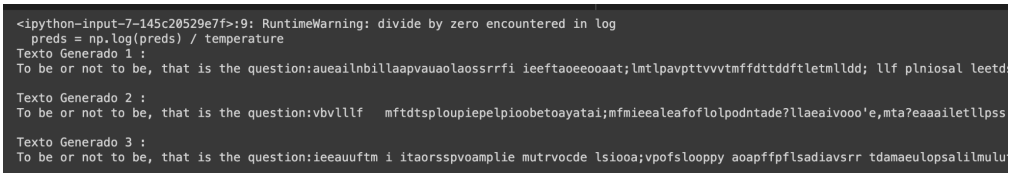

Figura 6. Generación de texto con seed distinto

Se puede pensar que el modelo no está generalizando bien, tiene una pérdida muy baja para los datos conocidos, pero al darle un nuevo seed no se comporta como se desearía o se esperaría. Sin embargo, considero que con los recursos utilizados se obtuvieron muy buenos resultados para el epoch 150, aunque las pruebas posteriores no fueron las mejores.

No se utilizaron técnicas como dropout por cuestiones de analizar el código original, pero como mejoras a futuro se propone emplear el dropout en las capas LSTM para reducir el sobreajuste, y usar data augmentation para crear más variaciones de texto a partir del original, sin embargo para este código solo se hicieron ciertas modificaciones al código original como el modificar el learning rate, la temperatura y el optimizador.

VIII. Conclusiones

La generación de texto con estilo de Shakespeare mediante Redes Neuronales Recurrentes (RNN) es un desafío complejo que requiere una combinación precisa de parámetros y entrenamiento. A través de la implementación de modelos LSTM (Long Short-Term Memory) en un entorno como Google Colab con GPU, se ha explorado la capacidad de estas redes para emular el estilo literario de Shakespeare.

Los resultados obtenidos en el proceso de entrenamiento y generación de texto demostraron una evolución significativa. Inicialmente, con un elevado valor de pérdida en el epoch 150, se observaron resultados poco coherentes y carentes de sentido en el texto generado. Sin embargo, con ajustes en hiperparámetros como el cambio del optimizador a Adam, la reducción del learning rate y la modificación de la temperatura para controlar la creatividad del modelo, se lograron mejoras notables.

A pesar de una pérdida reducida y resultados prometedores en el texto generado después del epoch 150, las pruebas posteriores revelaron dificultades en la generalización del modelo. Esto se tradujo en variaciones significativas en la coherencia y calidad del texto generado al utilizar diferentes semillas. Aunque se lograron avances notables en la generación de texto coherente y con palabras reconocibles, persisten desafíos en la consistencia y capacidad de generalización del modelo.

Para futuras mejoras, se propone la implementación de técnicas como dropout en las capas LSTM para mitigar el sobreajuste y el uso de data augmentation para

diversificar el conjunto de entrenamiento. Estas estrategias podrían contribuir a una mejora en la capacidad del modelo para generar texto coherente y consistente con el estilo de Shakespeare.

IX. Referencias

Kingma, D. P., & Ba, J. (2017). Adam: A method for stochastic optimization. In International Conference on Learning Representations (pp. 1-15). [Una guía completa para Adam y RMSprop Optimizer]