## RED NEURONAL

## Software básico.

|  |  |
| --- | --- |
| **software** | **Versión** |
| Windows | 10 |
| MacOS | 10.14 Mojave |
| Anaconda Navigator | 2.4.0 |
| Python | 3.8.17 |
| Tensorflow | 2.13.0 |
| Keras | 2.13.1 |
| Tensorflow\_text | 2.13.0 |
| Tensorflow\_datasets | 4.9.2 |
| pip | 23.2.1 |
| numpy | 1.24.3 |
| matplotlib | 3.7.2 |

## Hardware básico.

|  |  |
| --- | --- |
| **Hardware** | **Mínima requerida** |
| Memoria RAM | 16 Gz. |
| Espacio en disco | 10 Gz. |

## Modelo de la Red Neuronal para traducir de español a Kichwa

### Importación de módulos que serán usados

La implementación de una red neuronal para traducir texto del idioma Kichwa al español y viceversa requirió la utilización de diversas herramientas de procesamiento de lenguaje natural y aprendizaje automático. En este sentido, la importación de módulos es una tarea esencial para poder utilizar las funciones y clases que se necesitaron a lo largo del desarrollo del modelo. En este código, se presentan los principales módulos que fueron utilizados en la construcción de la red neuronal basada en el modelo Transformer. Entre ellos, se encontraban módulos para el preprocesamiento de texto, como la tokenización y el padding de secuencias, así como para la importación de datos de entrenamiento a través de la librería Tensorflow Datasets. Además, se incluyeron los módulos necesarios para la visualización de resultados a través de la librería Matplotlib.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

### Método para la lectura de los archivos con la data en español y Kichwa

Antes de poder desarrollar una red neuronal para traducir texto del idioma Kichwa al español y viceversa, fue necesario cargar los datos en la memoria de la computadora. En este caso, se prepararon archivos de texto que contenían frases y oraciones en español y Kichwa, respectivamente. Para leer y procesar estos archivos de manera eficiente, se definió el método "load\_data", que tomaba la ruta del archivo como entrada y devolvía una lista de oraciones en el idioma correspondiente. Al utilizar este método, se cargaban dos listas: "spanish\_vocab.txt" y "kichwa\_vocab.txt", que contenían las oraciones en español y Kichwa, respectivamente. A continuación, se podían usar esta información para entrenar y probar la red neuronal.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

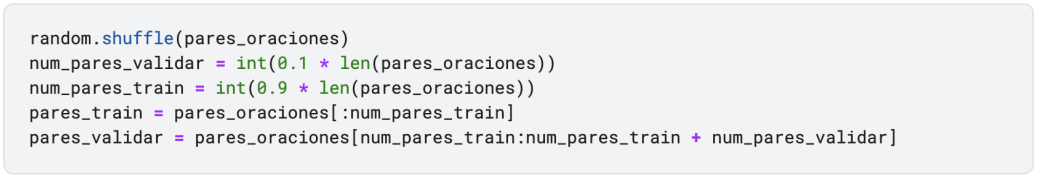
### Formar pares de oraciones de español y Kichwa

El siguiente código es una parte importante del proceso de preparación de datos para el entrenamiento de la red neuronal. El objetivo era formar pares de oraciones que incluían una oración en español y su correspondiente traducción al Kichwa. Para ello, se utilizó un bucle “for” que recorría cada oración en la lista de oraciones en español. En cada iteración, se agregaba la oración en español a la variable "spanish" y se incluía la correspondiente oración en Kichwa, con la etiqueta de inicio "[start]" y fin "[end]", a la variable "kichwa". Luego, se agregaba el par de oraciones formado por las dos variables a la lista "pares\_oraciones". Este proceso era crucial para el entrenamiento de la red neuronal, ya que proporcionaba los datos necesarios para que la red pudiera aprender a traducir del español al Kichwa y viceversa.



### Mezclar los pares de oraciones y obtener pares para entrenar y para validar

Este fragmento de código eran responsable de mezclar aleatoriamente los pares de oraciones y dividirlos en dos conjuntos, uno para entrenar y otro para validar el modelo. Primero, se usaba la función shuffle del módulo random para mezclar aleatoriamente los pares de oraciones contenidos en la lista pares\_oraciones. Luego, se definió el número de pares de oraciones que se usaban para validar el modelo y el número de pares que se utilizaban para entrenar el modelo. En este caso, se utilizó el 10% de los pares para validación y el 90% restante para entrenamiento. Finalmente, se crearon dos nuevas listas (pares\_train y pares\_validar) que contenían los pares de oraciones correspondientes a los conjuntos de entrenamiento y validación, respectivamente.



### Estandarizar las oraciones y convertir en tokens los datos de español y Kichwa

El siguiente paso en el proceso de desarrollo de la aplicación fue la estandarización y vectorización de las oraciones en español y Kichwa. Para ello, se definió una función de estandarización que convertía todo el texto en minúsculas y elimina la puntuación y los signos de interrogación. Luego, se creaba un objeto de vectorización de texto para la fuente de texto (español) y otro para el texto de destino (Kichwa) que convertía el texto en una secuencia de tokens. Estos objetos se adaptaban a los datos de entrenamiento y se configuraban con un máximo de tokens y una longitud máxima de secuencia. Una vez completado este proceso, se tenía una representación numérica de las oraciones en español y Kichwa que se podía utilizar para entrenar la red neuronal.

Texto

Descripción generada automáticamente

### Convertir en datasets los pares para entrenamiento y validación

Seguidamente, este código tomaba los pares de oraciones en español y Kichwa creados anteriormente y los convertía en datasets para entrenamiento y validación. Primero se definió el tamaño del lote (batch\_size=64) que se utilizaba para el entrenamiento. Luego, se definió la función set\_format para aplicar la vectorización de los datos utilizando los objetos TextVectorization creados previamente para cada idioma. Además, se ajustaba el formato de los datos para que la red neuronal pudiera entender la entrada y la salida. Posteriormente, se definía la función make\_dataset para crear un dataset a partir de los pares de oraciones en español y Kichwa, y se aplicaban las transformaciones necesarias a cada uno de los datasets. Finalmente, se devolvía el dataset creado para entrenamiento y validación.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

### Construcción del Encoder Codificador) y el Decoder (Decodificador) del Transformer



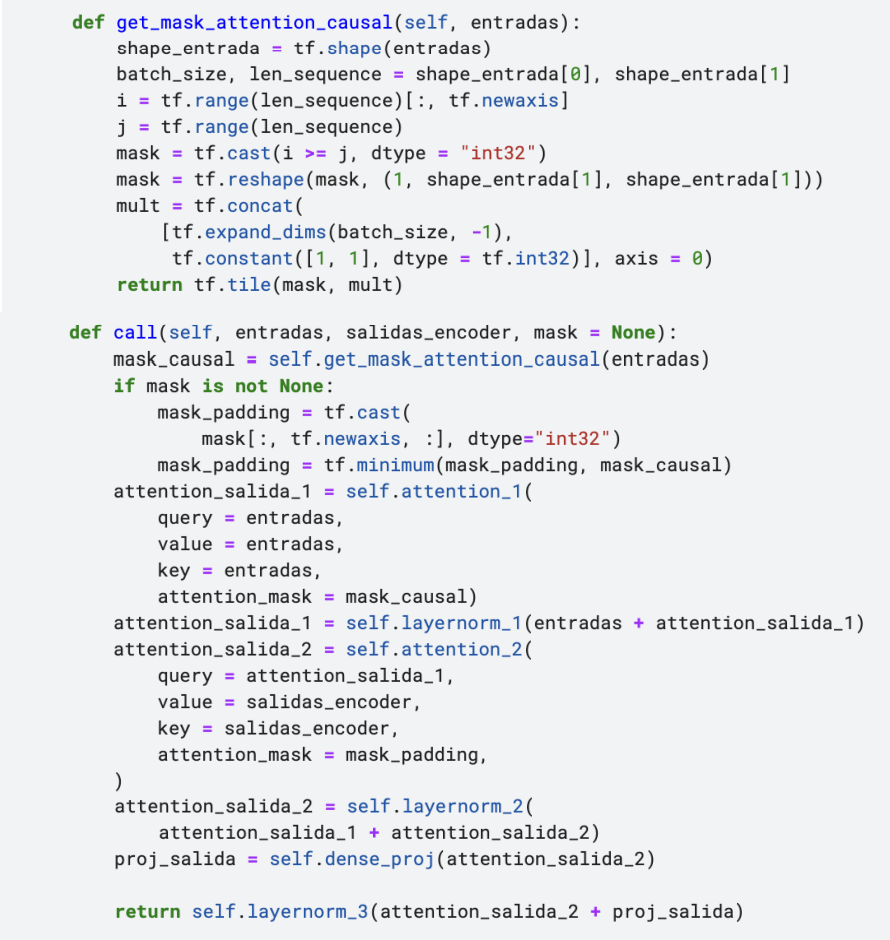
Seguidamente, el Decoder del Transformer era otra pieza fundamental en la arquitectura del Transformer, y se utilizaba para procesar la secuencia de salida. Al igual que el Encoder, el Decoder estaba compuesto por múltiples capas de atención y capas de feedforward. Sin embargo, a diferencia del Encoder, el Decoder también incluía una tercera capa de atención, que se utilizaba para conectar las entradas del Decoder con la salida del Encoder.

El Decoder tomaba como entrada la secuencia de salida y la salida del Encoder, y procesaba la secuencia de salida para generar una nueva secuencia de salida. En la implementación del código que mostró anteriormente, la clase DecoderTrans era la encargada de construir el Decoder. En el método “init”, se definió las capas de atención y feedforward, al igual que en el Encoder. Además, se definió una capa de atención adicional (attention\_2) para conectar las entradas del Decoder con la salida del Encoder.

Texto

Descripción generada automáticamente

El método get\_mask\_attention\_causal se utilizaba para crear una máscara de atención causal, que se empleaba para garantizar que el Decoder solo tuviera acceso a las posiciones previas en la secuencia de salida. Luego, en el método “call”, se utilizaban las capas de atención y feedforward para procesar la secuencia de salida y generar una nueva secuencia de salida.



### Construcción del Positional Encoding

Seguidamente, con este código se definió una capa de codificación posicional que se utilizaba en la arquitectura del Transformer. El objetivo del positional encoding era agregar información de posición a las representaciones de palabras en una secuencia de entrada, de modo que el modelo pudiera distinguir entre palabras en diferentes posiciones de la secuencia.

La capa de codificación posicional tomaba tres parámetros de entrada: dim\_entrada, que especificaba la dimensión de los vectores de entrada; dim\_salida, que especificaba la dimensión de los vectores de salida de la capa; y len\_sentences, que detallaba la longitud máxima de la secuencia de entrada.

En el constructor de la clase, se definió las dimensiones de entrada y salida de la capa, así como dos capas de embedding: una para los tokens de entrada y otra para las posiciones en la secuencia. La capa de embedding de tokens se inicializaba con los parámetros input\_dim y output\_dim, mientras que la capa de embedding de posición se inicializaba con len\_sentences y dim\_salida. La variable sequence\_length se definieron como len\_sentences.

En el método call, se calculaba el positional encoding para la entrada. Donde, primero se obtenía la longitud de la entrada con tf.shape(entradas)[-1], y luego se generaba un tensor de posiciones con tf.range(start=0, limit=limite, delta=1). Seguidamente, se aplicaba la capa de embedding de tokens y la capa de embedding de posición a las entradas y posiciones, respectivamente, y se sumaban los resultados para obtener la salida final.

El método compute\_mask se utilizaba para generar una máscara que se aplicaba a las entradas, de modo que las posiciones con un valor de entrada de cero no se tenían en cuenta en el cálculo de la atención. Finalmente, el método get\_config se utilizaba para obtener la configuración de la capa, que incluía los parámetros de entrada, la dimensión de salida y la longitud de la secuencia.





### Creación del modelo Transformer

Ahora que se ha definido la capa de positional encoding y las capas del encoder y decoder, es posible construir el modelo Transformer.

El modelo Transformer constaba de un encoder y un decoder conectados a través de un mecanismo de atención. El encoder procesaba la secuencia de entrada y produce una representación codificada, mientras que el decoder toma esta representación y genera la secuencia de salida.

Interfaz de usuario gráfica, Texto, Correo electrónico

Descripción generada automáticamente

### Compilación del modelo Transformer y entrenamiento

Mostrando que el modelo se está entrenando y mejorando la precisión a medida que pasan las épocas.

Texto

Descripción generada automáticamente con confianza media

### Almacenar archivos con el modelo de la red obtenido para ser usado por el server

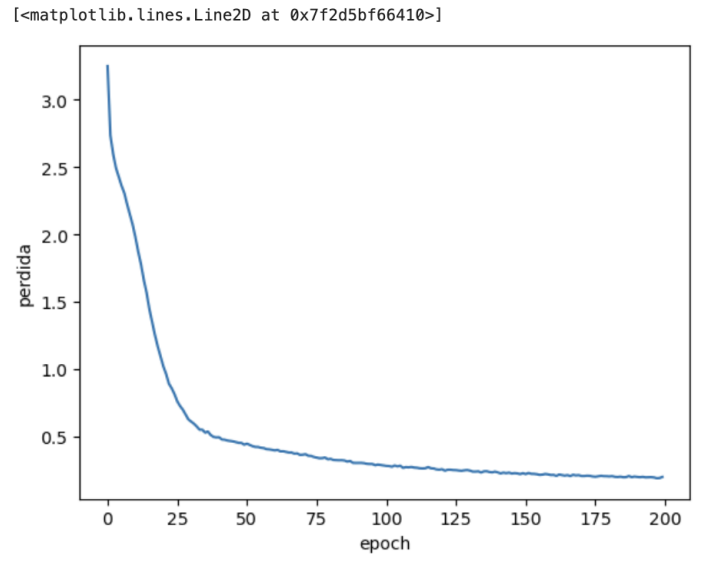
Texto

Descripción generada automáticamente

### Representación gráfica del marcador de pérdida en las epochs

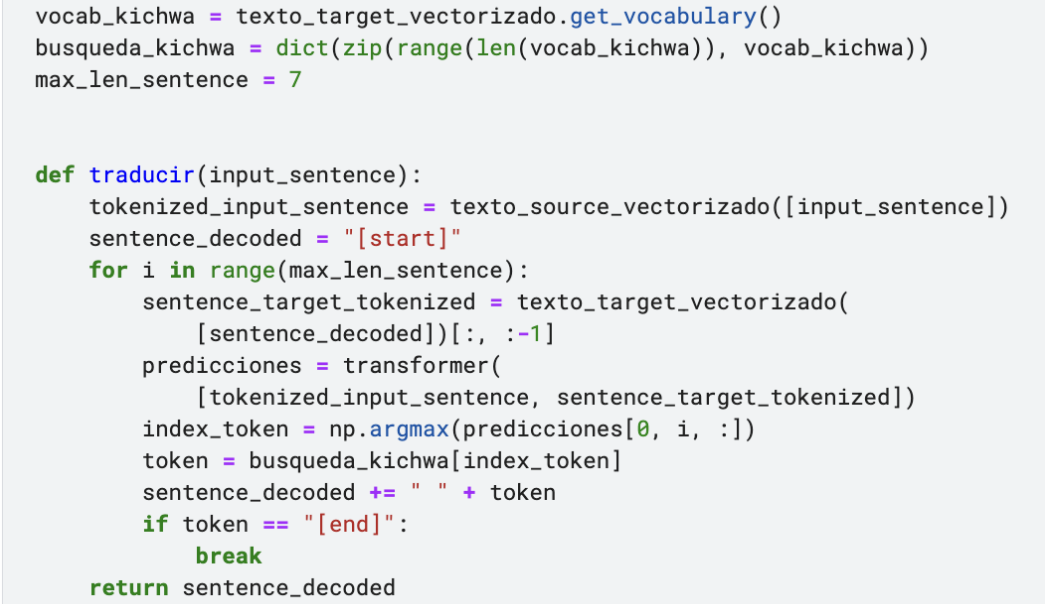
Imagen que contiene Gráfico

Descripción generada automáticamente

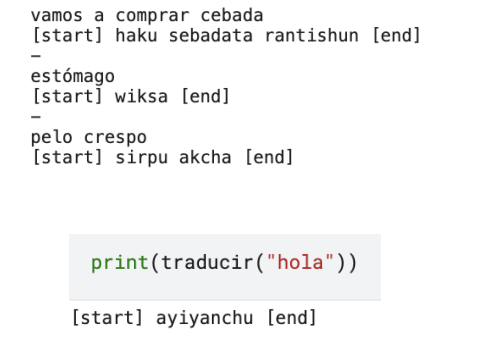


### Método para traducir de español a Kichwa

Este es el método para traducir de español a Kichwa. El modelo Transformer aprendido durante el entrenamiento se utilizaba para generar la traducción de una oración de entrada en español a una oración en Kichwa. La traducción se generaba mediante la predicción del modelo de cada palabra en la oración de salida, una palabra a la vez. El proceso de predicción se repetía hasta que se genera la palabra especial "[end]" o se alcanzaba la longitud máxima de la oración de salida (en este caso, 7).



### Ejemplos de traducción de español a Kichwa



## Modelo de la Red Neuronal para traducir de Kichwa a español

### Importación de módulos que serán usados

Para la traducción del Kichwa a español el proceso era muy similar al anterior, pero de forma inversa para realzar la funcionalidad tanto de un idioma hacia el otro.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

### Método para la lectura de los archivos con la data en Kichwa y español

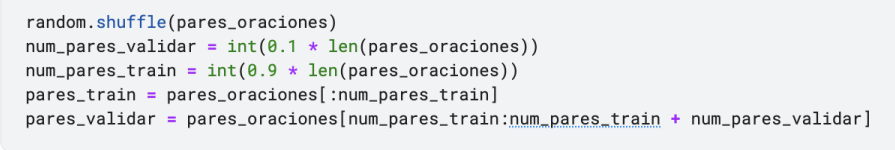
Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

### Formar pares de oraciones de Kichwa y español



### Mezclar los pares de oraciones y obtener pares para entrenar y para validar



### Estandarizar las oraciones y convertir en tokens los datos de Kichwa y español



### Convertir en datasets los pares para entrenamiento y validación

Texto

Descripción generada automáticamente

### Construcción del Encoder del Transformer







### Construcción del Positional Encoding

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

### Creación del modelo Transformer



### Compilación del modelo Transformer y entrenamiento

Texto

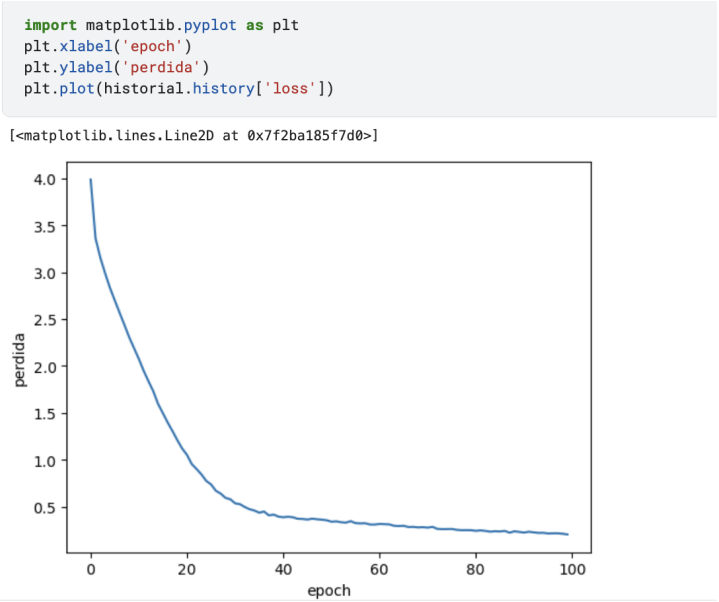
Descripción generada automáticamente

### Almacenar archivos con el modelo de la red obtenido para ser usado por el server

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

### Representación gráfica del marcador de pérdida en las epochs



### Método para traducir de Kichwa a español

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

### Ejemplos de traducción de Kichwa a español

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente