Universidad Zaragoza

Facultad de Ciencias
Universidad Zaragoza

UNIVERSIDAD DE ZARAGOZA

MÁSTER UNIVERSITARIO EN MODELIZACIÓN E INVESTIGACIÓN MATEMÁTICA, ESTADÍSTICA Y COMPUTACIÓN.

Trabajo de Fin de Máster

Construcción de un sistema de Inteligencia Artificial para la transcripción de audio a texto para la gestión de incidencias

Aurea López Rodríguez

Directores del trabajo:
Javier Díez & Beatriz Lacruz

septiembre de 2023

Abstract

In the last years, the field of artificial intelligence has experienced a significant growth, matching with advancements in the field of deep learning. This evolution has allowed an expansion of the techniques and algorithms employed in the Automatic Speech Recognition (ASR).

This work focuses on the development of an artificial intelligence system designed for precise audio-to-text transcription, with the intention of applying it to incident management. In particular, we will analyze models that use Bidirectional Convolutional Recurrent Neural Networks (BCRNN).

The research begins with an analysis of methods employed to extract features from audio signals, including Mel-Frequency Cepstral Coefficients (MFCC) and spectrograms. These features, coupled with literal transcriptions of the audios, will serve as input data for our network.

Next, the fundamental components of the BCRNN are examined in detail, analyzing the elements of convolutional and bidirectional recurrent layers of the network, explaining their role in sequence data processing.

Once the elements that constitute these models are established, the learning process is explained through the optimization of a loss function that will allow adjusting the network parameters to minimize error. In our case, this loss function is Connectionist Temporal Classification (CTC), which is combined with the Back-Propagation Through Time (BPTT) algorithm and the Stochastic Gradient Descent (SGD) optimization algorithm.

To finish this theoretical section, the two most commonly used metrics for ASR model validation are defined: the Word Error Rate (WER) and the Character Error Rate (CER).

The final part of this work applies the previously described theoretical concepts to the study of the viability of a project focused on incident type detection.

The project has been developed using Jupyter Notebook and is presented with a detailed explanation of each step taken during its execution. It begins with the description of the process of dataset construction. Then, several models that have been developed for this project are presented, providing a description of their architecture and the results achieved in each case.

Within this section, special attention is given to the final model, which is specifically adapted for this project and is based on a BCRNN architecture. The model compilation incorporates the CTC loss function and the SGD optimizer. The results obtained with this model have been highly satisfactory, getting an accuracy of 97,48 % in terms of characters.

Finally, a conclusions section is presented with potential lines of future work to further improve the developed model.

Resumen

En los últimos años, el campo de la inteligencia artificial ha experimentado un crecimiento significativo, impulsado por los avances en el campo del *deep learning*. Esta evolución ha permitido una ampliación significativa de las técnicas y algoritmos empleados en el ámbito del reconocimiento automático del habla (ASR).

Este trabajo se centra en la creación de un sistema de inteligencia artificial para lograr una transcripción precisa de audio a texto, con la finalidad de aplicarlo en la gestión de incidencias. En particular, se analizan modelos que utilizan redes neuronales bidireccionales convolucionales recurrentes (BCRNN).

La investigación comienza con un análisis exhaustivo de los métodos para extraer características de las señales de audio, incluyendo los coeficientes cepstrales de Mel (MFCC) y los espectrogramas. Estas características, junto con las transcripciones textuales de los audios, servirán como datos de entrada para nuestra red.

A continuación, se examinan en detalle las componentes fundamentales de una BCRNN, desglosando los elementos que conforman las capas convolucionales y las capas recurrentes bidireccionales de la red, explicando su funcionamiento en el procesamiento de secuencias de datos.

Una vez establecidos los elementos que componen estos modelos, se explora el proceso de aprendizaje mediante la optimización de una función de pérdida que permitirá ajustar los parámetros de la red para minimizar el error. En nuestro caso, esta función de pérdida es la clasificación temporal de conexiones (CTC), que se combina con el algoritmo de retropropagación a través del tiempo (BPTT) y el algoritmo de optimización de descenso del gradiente estocástico (SGD).

En la culminación de esta sección teórica, se definen las dos métricas más utilizadas para la validación de modelos ASR: el *Word Error Rate* (WER) y el *Character Error Rate* (CER).

La última parte de este trabajo aplica las herramientas teóricas previamente descritas al estudio de la viabilidad de un proyecto enfocado en la detección del tipo de incidencia.

El proyecto se ha desarrollado utilizando *Jupyter Notebook* y se presenta detallando cada uno de los pasos realizados durante su ejecución. Se inicia con la descripción del proceso de construcción del conjunto de datos. Luego, se presentan los diversos modelos que se han ajustado para este proyecto, describiendo en profundidad la arquitectura y los resultados obtenidos en cada caso.

Dentro de esta sección, se brindará particular atención al modelo final, que se adapta específicamente a este proyecto y se basa en una red BCRNN. La compilación del modelo incorpora la función de pérdida CTC y el optimizador SGD. Los resultados alcanzados con este modelo han sido bastante buenos, obteniendo una precisión del 97,48 % en términos de caracteres.

Finalmente, se proporciona una sección de conclusiones, junto con posibles líneas de trabajo futuro para continuar mejorando el modelo desarrollado.

Índice general

Abstract III

Resumen V

1. Introducción 1
- 1.1. Motivación y objetivos . . . 1
- 1.2. Entorno tecnológico . . . 3
- 1.3. Estructura del trabajo . . . 3

2. Reconocimiento automático del habla (ASR) 5
- 2.1. Estado del arte . . . 5
- 2.2. Extracción de características . . . 6
  - 2.2.1. Fragmentación . . . 8
  - 2.2.2. Ventaneo . . . 8
  - 2.2.3. Conversión a la escala de Mel . . . 9
  - 2.2.4. Cálculo del logaritmo y normalización . . . 10
  - 2.2.5. Transformada de coseno discreta (DCT) . . . 11

3. Redes Neuronales para el ASR 13
- 3.1. Redes Neuronales Bidireccionales Convolucionales Recurrentes . . . 13
  - 3.1.1. Capa convolucional . . . 14
  - 3.1.2. Capas recurrentes bidireccionales . . . 17
- 3.2. Aprendizaje de la red . . . 22
  - 3.2.1. Función de perdida . . . 22
  - 3.2.2. Algoritmo de retropropagación a través del tiempo (BPTT) . . . 27
  - 3.2.3. Descenso del gradiente estocástico (SGD) . . . 29
  - 3.2.4. Funciones de activación en la etapa de aprendizaje . . . 30
  - 3.2.5. *Batch normalization* . . . 33
  - 3.2.6. Sobreajuste . . . 33
- 3.3. Validación de la red . . . 34
  - 3.3.1. *Word Error Rate* (WER) . . . 34
  - 3.3.2. *Character Error Rate* (CER) . . . 35

4. Creación y evaluación del proyecto 37
- 4.1. Elaboración del conjunto de datos . . . 37
- 4.2. Creación del modelo . . . 39
  - 4.2.1. Primer modelo . . . 40
  - 4.2.2. Segundo modelo . . . 43
  - 4.2.3. Modelo final . . . 44
- 4.3. Conclusiones y trabajo futuro . . . 47

Capítulo 1

Introducción

1.1. Motivación y objetivos

En el ámbito de la gestión de incidencias, la capacidad de capturar y comprender rápidamente la información es crucial para una respuesta efectiva y oportuna. Sin embargo, el procesamiento manual de grandes volúmenes de información, como la transcripción de audio a texto, puede ser lento y propenso a errores. Aquí es donde la Inteligencia Artificial (IA) y el reconocimiento automático del habla (ASR) desempeñan un papel fundamental.

El propósito de este trabajo es construir un sistema de Inteligencia Artificial destinado a la transcripción de audio a texto, una iniciativa propuesta por la empresa Efor. Efor es una empresa con sede en Aragón que forma parte del grupo Integra dedicada a dar servicios y soluciones tecnológicas para la gestión, comunicación y marketing de las empresas. El objetivo final de esta propuesta es emplear este sistema en un futuro para optimizar la gestión de incidencias. El sistema propuesto aprovecha las capacidades avanzadas de procesamiento del lenguaje natural para automatizar el proceso de transcripción, permitiendo una mayor eficiencia y precisión en la captura de información relevante.

Para alcanzar este objetivo, se utilizarán técnicas de reconocimiento de voz y algoritmos de aprendizaje automático que transformarán automáticamente las grabaciones de audio en texto legible. El objetivo principal de este trabajo es eliminar la necesidad de que el personal tenga que escuchar y transcribir manualmente las grabaciones, lo que resultará en un ahorro de tiempo significativo y reducirá los posibles errores humanos.

Para llevar a cabo esta tarea, es necesario contar con un algoritmo capaz de procesar y analizar audios para que puedan ser interpretados por un ordenador. En este sentido, el campo del procesamiento de audio y el ASR juegan un papel fundamental. Antes de profundizar en los modelos que utilizaremos en este trabajo, es importante comprender cómo el ordenador percibe y procesa las señales de audio, ya que estas serán los datos utilizados para obtener las entradas a la red.

Para un ordenador, una señal de audio, como la que se muestra en la Figura 1.1, es una función que asigna valores complejos a una secuencia de puntos en el dominio temporal, es decir, $x : \mathbb{Z}^n \longrightarrow \mathbb{C}$. En el contexto del procesamiento de señales de audio, la transformada de Fourier y la Transformada de Coseno Discreta (DCT) son dos técnicas ampliamente utilizadas para analizar señales en el dominio de la frecuencia.

La transformada de Fourier es una herramienta fundamental para obtener la representación en el dominio frecuencial de una señal. Esta transformada descompone la señal en componentes de diferentes frecuencias, lo que proporciona información sobre la distribución de energía en el espectro de frecuencias.

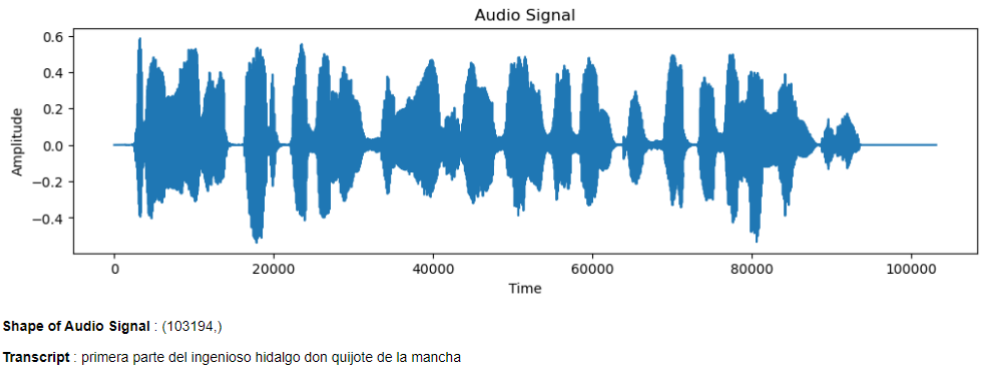

Figura 1.1: Señal de audio correspondiente a la frase “Primera parte del ingenioso hidalgo don quijote de la mancha”.

Matemáticamente, la transformada de Fourier de una señal x se define como la suma ponderada de las componentes exponenciales complejas correspondientes a cada frecuencia ω. Formalmente, se calcula como:

$$X(\omega) = \sum_{n=-\infty}^{+\infty} x(n)e^{-in\omega}, \ \omega \in \mathbb{R}.$$

Esta expresión muestra cómo cada instante de tiempo n de la señal x contribuye a la frecuencia ω a través de la componente exponencial compleja $e^{-in\omega}$.

Es importante destacar que, para realizar cálculos eficientes en el procesamiento de señales, se utiliza la transformada rápida de Fourier (FFT). La FFT es un algoritmo que aprovecha la estructura de la transformada de Fourier y utiliza técnicas de fragmentación y ventaneo de la señal para reducir la complejidad computacional del cálculo. Esto permite realizar análisis de las componentes frecuenciales presentes en una señal de audio de manera rápida y eficiente.

La información obtenida a través de la transformada de Fourier y la FFT es fundamental para comprender y caracterizar diferentes aspectos del audio, como el tono, el ritmo y el contenido espectral. Estos conocimientos son de gran importancia en el procesamiento de señales de audio, ya que permiten realizar tareas como la detección de frecuencias dominantes, el análisis de patrones temporales y la identificación de características relevantes en la señal.

Por otro lado, la Transformada de Coseno Discreta (DCT) también se utiliza en el procesamiento de señales para transformar una señal del dominio del tiempo al dominio de la frecuencia. Al igual que la transformada de Fourier, la DCT descompone una señal en una combinación de componentes de frecuencia. Sin embargo, se utiliza específicamente en aplicaciones donde la señal de entrada es discreta en el tiempo y se busca una representación compacta en términos de coeficientes de coseno.

La DCT se basa en la descomposición de una señal discreta en una serie de cosenos ortogonales. Estos cosenos se definen por su frecuencia y posición en el tiempo, y la DCT calcula los coeficientes que representan la contribución de cada coseno en la señal original. Matemáticamente, la DCT se define como:

$$X(\omega) = \sum_{n=0}^{N-1} x(n) \cdot \cos\left(\frac{\pi}{N} \cdot \left(n + \frac{1}{2}\right) \cdot k\right), \quad k = 0, 1, 2, ..., N-1,$$

donde N es la longitud de la señal de entrada.

La DCT es ampliamente utilizada en aplicaciones de compresión de audio, donde se utilizan los coeficientes para reducir la redundancia y la información no relevante en la señal, logrando una representación más compacta y eficiente del audio.

En resumen, el procesamiento de señales de audio por parte del ordenador implica considerar las señales como una secuencia de valores numéricos en el dominio temporal. Mediante técnicas como la transformada de Fourier y la DCT, se obtienen representaciones en el dominio frecuencial que permiten analizar y caracterizar diferentes aspectos del audio.

En este contexto, las redes neuronales han surgido como una solución prometedora para la transcripción de audio a texto legible. Estas redes, entrenadas en conjuntos de datos extensos, tienen la capacidad de aprender patrones complejos en el habla y generar transcripciones precisas. Al aprovechar el poder de las redes neuronales, se espera mejorar la eficiencia y calidad en la captura de información relevante, lo que a su vez se espera que contribuya a una gestión más efectiva de las incidencias.

Con el fin de construir un sistema de IA para la transcripción de audio a texto aplicable a la gestión de incidencias, en este trabajo exploraremos y desarrollaremos modelos y algoritmos basados en redes neuronales, aprovechando las técnicas avanzadas del ASR. Estos avances tecnológicos tienen el potencial de revolucionar la forma en que se manejan y procesan las señales de audio, mejorando la calidad en la captura de información crítica.

1.2. Entorno tecnológico

Los entornos de trabajo elegidos para llevar a cabo el entrenamiento de la red y la creación del algoritmo capaz de transcribir audio a texto han sido los siguientes:

- *Tensorflow* [26]: Es una biblioteca de código abierto para aprendizaje automático desarrollada por Google, que proporciona una API de *Python*.
- *Keras* [13]: Es una biblioteca diseñada para la experimentación con redes de *deep learning* escrita en *Python*. Puede ejecutarse sobre el entorno de *Tensorflow*.

Para la implementación de estos paquetes y librerías hemos utilizado *Jupyter Notebook* [15], en un entorno de Anaconda. *Jupyter Notebook* es una aplicación web que permite crear cuadernos donde insertar código, material visual y texto narrativo. Esta aplicación tiene un amplio rango de aplicaciones, que incluyen la simulación numérica, el modelado estadístico y, en nuestro caso, el aprendizaje automático.

Además, esta aplicación cuenta con un núcleo encargado de ejecutar el código del cuaderno. *Jupyter* soporta más de 40 lenguajes de programación como R, *Scala* o *Python* que será el que usemos en nuestro caso. Una de las principales razones por las que hemos elegido esta aplicación como entorno para ejecutar el código es que permite introducir comentarios en lenguaje LaTeX lo que facilita la introducción de fórmulas matemáticas en celdas adjuntas al código, así como la creación de pequeños informes.

Aparte de los entornos de programación, para el desarrollo del trabajo se ha utilizado un portátil cedido por la empresa Efor, que tiene instalado un sistema operativo Windows 10 Pro de 64 bits. El portátil cuenta con un procesador Intel(R) Core(TM) i5-1145G7 y 16GB de memoria RAM.

1.3. Estructura del trabajo

Esta memoria está organizada en tres capítulos, además de este primero, que consiste en una breve introducción a la temática del trabajo.

En los dos siguientes capítulos, se abordará en detalle la teoría relacionada con el tipo de redes neuronales utilizadas en el campo del reconocimiento automático del habla (ASR), así como la extracción de características necesarias para la obtención de los datos de entrada. Estos datos de entrada están formados por las características obtenidas, como los MFCC o los valores de los espectrogramas, a partir de una serie de audios que contienen frases del libro "El Quijote". Asociadas a estas características, también se consideran las transcripciones reales del audio correspondiente, es decir, la representación textual de las frases habladas en cada archivo de audio. En particular, en el capítulo 2 se profundizará en las técnicas empleadas para extraer estas características de los datos de entrada, lo cual es crucial para el correcto funcionamiento de las redes neuronales en el proceso del ASR. La combinación de estas características y las transcripciones reales permitirá entrenar y validar los modelos, facilitando la comprensión y análisis del habla humana mediante técnicas computacionales avanzadas. Y en el capítulo 3, se describirán los elementos que conforman las redes neuronales utilizadas en el trabajo, junto con el proceso de entrenamiento y validación del modelo. Se explorará en detalle la arquitectura de la red y los parámetros utilizados para lograr resultados óptimos.

Finalmente, el último capítulo, se enfocará en la implementación práctica de los conocimientos teóricos adquiridos, con el objetivo de transcribir audio a texto. Se detallará el proceso de obtención del conjunto de datos utilizado como entrada para la red neuronal. Posteriormente, se presentarán los resultados obtenidos a partir del entrenamiento de diferentes modelos y, por último, se expondrán conclusiones relevantes que contemplen posibles modificaciones y mejoras del programa.

A lo largo de este trabajo, se busca presentar un enfoque claro y detallado sobre el ASR utilizando redes neuronales, cuyo desarrollo e investigación han sido elaboración propia. El objetivo es proporcionar una base sólida para comprender y aplicar estas técnicas en futuros proyectos relacionados con el procesamiento del lenguaje natural y el ASR. Cabe destacar que todas las Figuras presentadas en este trabajo son de elaboración propia.

Capítulo 2

Reconocimiento automático del habla (ASR)

2.1. Estado del arte

El reconocimiento automático del habla (ASR) es una tecnología que convierte el habla humana en texto de manera automatizada. Ha sido ampliamente investigada y desarrollada en las últimas décadas, y se ha aplicado en diversos campos como la telefonía, los asistentes virtuales, la transcripción y la traducción automática, entre otros. En este capítulo, se presenta una revisión exhaustiva del estado del arte del reconocimiento automático del habla, destacando los avances más significativos y las técnicas más utilizadas.

El desarrollo del ASR se remonta a los primeros intentos de Alexander Graham Bell en la década de 1870, cuando descubrió la posibilidad de convertir ondas sonoras en impulsos eléctricos. Sin embargo, los primeros sistemas de reconocimiento del habla eran limitados y se centraban en la identificación de dígitos telefónicos de calidad. Con el tiempo, se han logrado avances significativos en términos de precisión y aplicaciones del ASR.

Durante las primeras etapas de desarrollo del ASR, se utilizaron enfoques clásicos basados en modelos ocultos de Markov (HMM) y modelos acústicos-lingüísticos [7, 2, 17]. Estos enfoques utilizaban modelos probabilísticos para representar las características acústicas del habla y los patrones lingüísticos. A pesar de sus limitaciones, estos enfoques lograron resultados prometedores y sentaron las bases para el desarrollo posterior del ASR.

En las últimas décadas, los avances en el procesamiento de señales, la inteligencia artificial y el aprendizaje automático han impulsado de manera significativa el desarrollo del ASR. Estos avances han permitido mejorar la precisión y la eficiencia de los sistemas ASR, así como expandir sus aplicaciones en diversos campos.

Uno de los principales avances ha sido la utilización de enfoques basados en redes neuronales, que han demostrado un rendimiento superior en comparación con los enfoques clásicos. Entre estos enfoques se encuentran las redes neuronales convolucionales (CNN) [1], las redes neuronales recurrentes (RNN) [28], las redes neuronales convolucionales recurrentes (CRNN) [3] y las redes bidireccionales convolucionales recurrentes (BCRNN) [31]. Estas arquitecturas de red han demostrado ser altamente efectivas en el procesamiento de secuencias de audio, superando las limitaciones de los modelos basados en HMM en términos de precisión y capacidad de generalización.

En nuestro trabajo, nos centramos en la utilización de redes neuronales del tipo BCRNN. Las BCRNN son una arquitectura de red que combina elementos de redes convolucionales y redes recurrentes, apro-

vechando las ventajas de ambas para mejorar la precisión y el rendimiento en el reconocimiento del habla.

Las redes neuronales convolucionales (CNN) son especialmente adecuadas para el procesamiento de datos de tipo imagen, ya que son capaces de aprender características locales y patrones espaciales a través de capas convolucionales. Estas características se capturan mediante filtros convolucionales que se deslizan sobre la secuencia temporal de la señal de audio.

Por otro lado, las redes neuronales recurrentes (RNN) son capaces de modelar secuencias de manera efectiva, ya que utilizan conexiones recurrentes que les permiten tener una memoria a largo plazo. En el contexto del ASR, las RNN son utilizadas para capturar la estructura temporal y las dependencias a largo plazo presentes en las señales de audio. Estas redes son especialmente útiles para el reconocimiento del habla, donde la información contextual y la correlación temporal son cruciales.

La combinación de las características aprendidas por las capas convolucionales con la capacidad de modelado secuencial de las capas recurrentes permite a las BCRNN capturar tanto información local como dependencias a largo plazo en las señales de audio. Esto resulta en una mejora significativa en la precisión y la capacidad de generalización del modelo de reconocimiento automático del habla.

Además, el uso de modelos de lenguaje basados en transformadores ha demostrado ser otro avance importante en el campo del ASR [12, 11]. Estos modelos utilizan arquitecturas de red basadas en la atención para capturar patrones y dependencias a largo plazo en el habla. Los transformadores han demostrado ser altamente efectivos para mejorar la coherencia y la fluidez de las transcripciones generadas por los sistemas ASR. Al incorporar estos modelos de lenguaje basados en transformadores en la arquitectura de las BCRNN, podemos mejorar aún más la calidad y la comprensión del texto transcrito.

Si nos enfocamos en las redes neuronales, existen decisiones clave que deben tomarse, como seleccionar la arquitectura de la red y determinar las características que se utilizarán como entrada. La extracción de características es un proceso fundamental que busca transformar los datos de entrada en un conjunto de características que capturen las propiedades relevantes y distintivas de los datos. Esto tiene como objetivo mejorar la eficiencia y precisión de los algoritmos de aprendizaje automático.

En nuestro caso particular, en lugar de utilizar directamente la forma de onda de la señal de audio como entrada de la red, se extraen características numéricas que contienen información relevante para el análisis del habla. Entre las técnicas más utilizadas se encuentran los espectrogramas y los coeficientes cepstrales de frecuencia Mel (MFCC). Estas técnicas permiten representar de manera efectiva la información acústica del habla de forma más compacta y significativa.

Además de los espectrogramas y los MFCC, existen otras técnicas de extracción de características utilizadas en el ASR, como los coeficientes lineales predictivos (LPC) o las frecuencias espectrales de línea (LSF), entre otros [14].

2.2. Extracción de características

La selección adecuada de las características a utilizar en el ASR depende del contexto y los objetivos específicos del sistema. Es crucial considerar las características que se adapten mejor al tipo de datos y la tarea en cuestión, ya que esto puede tener un impacto significativo en la precisión y eficiencia del sistema.

En nuestro trabajo, hemos optado por utilizar espectrogramas y MFCC como características de entrada. A continuación, explicaremos la obtención de cada uno de ellos a partir de los datos de entrada. Los pasos realizados para obtener estas características se detallan en las Figuras 2.1 y 2.3.

El espectrograma es una representación visual del espectro de frecuencia de una señal de audio a lo largo del tiempo, como se muestra en la Figura 2.2. Se obtiene mediante la aplicación de la Transformada de Fourier de Tiempo Corto (STFT), que es una extensión de la FFT detallada en la sección 1.1, diseñada para abordar señales no estacionarias. El proceso del espectrograma divide la señal en segmentos de tiempo y muestra la distribución de energía en diferentes frecuencias, permitiendo identificar cambios y patrones en el audio.

Figura 2.1: Pasos para la obtención de un espectrograma.

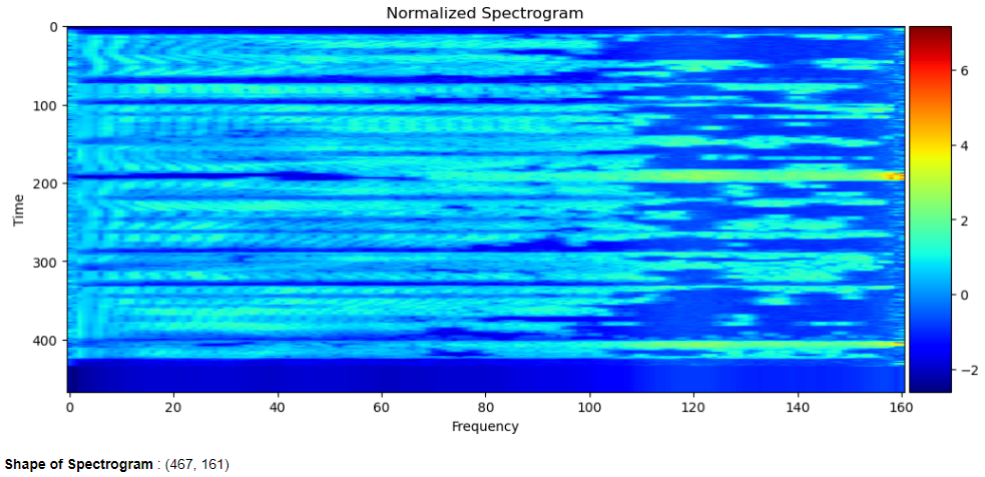

Figura 2.2: Ejemplo de espectrograma del audio representado en la Figura 1.1.

Por otro lado, los MFCC son un conjunto de características que representan la amplitud del espectro del habla de manera compacta, como se muestra en la Figura 2.4. Por ello, se han vuelto una de las técnicas de extracción de características más usada en reconocimiento del habla. Estos coeficientes se calculan mediante la transformada de coseno discreta de los valores logarítmicos del espectrograma de Mel.

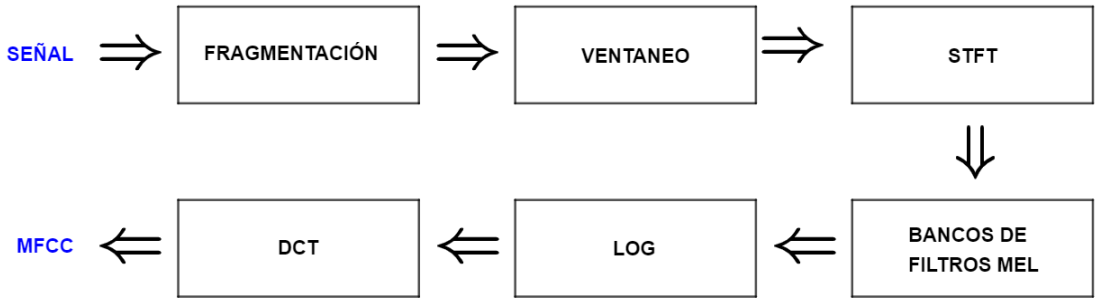

Figura 2.3: Pasos para la obtención de los coeficientes cepstrales de frecuencia Mel (MFCC).

Es importante destacar que, antes de aplicar estas técnicas de extracción de características, es necesario realizar etapas de preprocesamiento, como la fragmentación y ventaneo de la señal de audio, que permiten dividir la señal en segmentos más manejables y representativos para su posterior análisis. Analizamos con más detalle cada una de estas etapas.

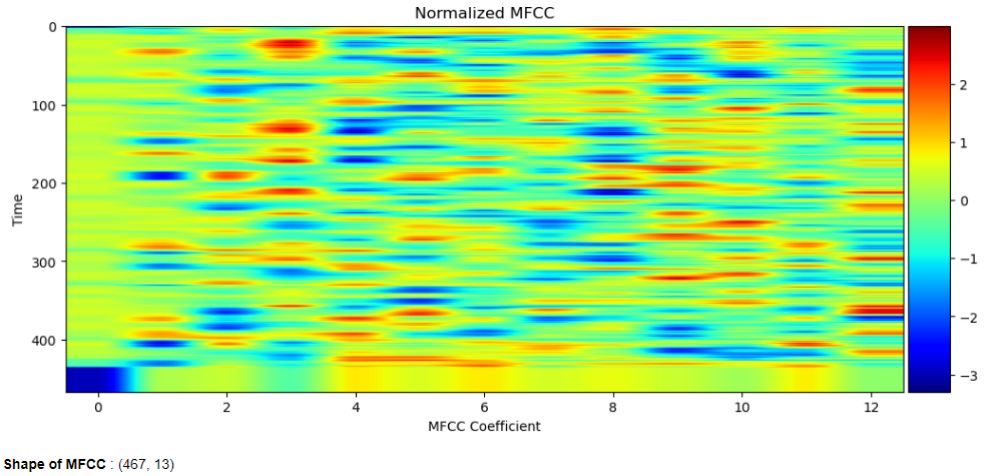

Figura 2.4: Ejemplo de representación del espectrograma de Mel del audio representado en la Figura 1.1.

2.2.1. Fragmentación

La fragmentación implica dividir la señal de audio en secciones cortas, ya que el espectro de una señal de audio cambia rápidamente a lo largo del tiempo. Aunque el habla es considerada una señal no estacionaria, al fragmentarla en secciones cortas se puede asumir una "cuasi-estacionalidad", lo que significa que sus propiedades estadísticas son constantes dentro de cada sección. Esto permite aplicar la transformada de Fourier para obtener el espectro de la señal.

2.2.2. Ventaneo

El ventaneo se utiliza para eliminar las posibles discontinuidades entre el final y el principio de una señal. Esto evita el fenómeno conocido como fenómeno de Gibbs [18]. Para minimizar este fenómeno, se aplican funciones de ventana a los diferentes fragmentos de la señal. Estas funciones de ventana suavizan los bordes de la señal, asegurando una transición continua entre los fragmentos. Al aplicar el ventaneo, es necesario tomar decisiones respecto al tamaño de la ventana, la superposición entre ventanas y el tipo de función de ventana a utilizar.

El tamaño de la ventana debe ser limitado a unos pocos milisegundos para poder asumir la cuasi-estacionalidad. Un tamaño de ventana más pequeño permite una suposición de estacionalidad más confiable, pero también resulta en una menor resolución en frecuencia y un mayor costo computacional debido al procesamiento de más ventanas. Por lo general, el tamaño típico para el procesamiento de señales de habla oscila entre 20 y 40 milisegundos [16]. El desplazamiento entre ventanas sucesivas suele ser ligeramente inferior a la mitad del tamaño de la ventana para lograr un solapamiento parcial y compensar la atenuación experimentada en los extremos al aplicar la función de ventana.

Existen diversas funciones de ventana disponibles para el procesamiento de señales. Estas funciones proporcionan información valiosa sobre la resolución en frecuencia de la señal y las perturbaciones que se producen en el dominio de frecuencia.

La ventana rectangular es la función de ventana más básica y simple. A diferencia de otras funciones de ventana más avanzadas, la ventana rectangular no resuelve el problema del fenómeno de Gibbs y es equivalente a no aplicar ninguna función de ventana en absoluto. Se define como:

$$w(n) = \begin{cases} 1, \text{ si } 0 \leq n \leq N-1, \\ 0, \text{ otro caso }, \end{cases} \tag{2.1}$$

donde N es el tamaño de la ventana.

Una de las funciones de ventana más utilizadas en el análisis de señales de audio es la ventana de Hamming. Esta reduce el fenómeno de Gibbs al atenuar los extremos de la señal. Matemáticamente se expresa como:

$$w(n) = a_0 - (1 - a_1)\cos\left(\frac{2\pi n}{N-1}\right),\ 0 \leq n \leq N-1, \tag{2.2}$$

donde N es el tamaño de la ventana y a_0 y a_1 son coeficientes que se pueden ajustar para mejorar la resolución espectral.

En este trabajo, se utilizará un caso particular de la ventana de Hamming con coeficientes específicos, conocida como ventana de Hann, que toma como coeficientes $a_0 = \frac{1}{2} = a_1$. Esta elección se basa en consideraciones de rendimiento y propiedades espectrales. Al utilizar estos coeficientes, se obtiene una ventana suavizada con una buena capacidad de reducción del fenómeno de Gibbs ofreciendo un rendimiento satisfactorio en el análisis de señales de habla.

Además de las ventanas mencionadas, existen otras ventanas populares que pueden ser de interés en el procesamiento de señales, como la ventana de Blackman y la ventana de Kaiser [25].

Después de aplicar la función de ventana a una señal, se procede a aplicar la transformada de Fourier a cada fragmento ventaneado para obtener su representación en el dominio de la frecuencia. Comúnmente, se utiliza la transformada de Fourier de tiempo corto (STFT) para calcular eficientemente la transformada de Fourier discreta de una señal no estacionaria.

Para obtener el espectrograma, que muestra la variación de la energía espectral en función del tiempo y la frecuencia, se utilizan los coeficientes de amplitud obtenidos mediante la STFT. Sin embargo, la obtención de los MFCC requiere algunos pasos adicionales que se explican a continuación.

2.2.3. Conversión a la escala de Mel

La escala de Mel es una escala perceptual de frecuencia que se ajusta mejor a la forma en la que el oído humano percibe los cambios en la frecuencia. Esto permite que los filtros de Mel cubran de manera adecuada el rango de frecuencias audibles. Como se muestra en la Figura 2.5 (derecha), la relación entre la frecuencia, f y la frecuencia en la escala de Mel, m, se define mediante la siguiente expresión matemática:

$$m = 2595\ \log(1 + \frac{f}{700}). \tag{2.3}$$

Para obtener los MFCC, se aplican filtros de Mel a los resultados de la STFT. Estos filtros de Mel son una serie de filtros triangulares que se superponen en el dominio de la frecuencia, como los que se muestran en la Figura 2.5 (izquierda). Cada filtro representa una banda de frecuencia específica en la escala de Mel. La ubicación y forma de los filtros de Mel se eligen de manera que cubran adecuadamente el rango de frecuencias audibles y se ajusten a la respuesta auditiva humana.

El diseño de los bancos de filtros comienza seleccionando las frecuencias mínima y máxima que se consideran útiles. Para señales de voz, como en este caso, se elige este rango para capturar el tono y los armónicos más importantes de la voz humana, típicamente entre 300 Hz y 8 kHz. Luego, se divide el espacio en Mels a partes iguales según el número de bancos de filtros deseado. Es importante tener en cuenta que la escala de Mel distingue mejor las frecuencias bajas que las altas, como se muestra en la Figura 2.5 (derecha). Por lo tanto, los bloques en hercios serán más anchos a altas frecuencias.

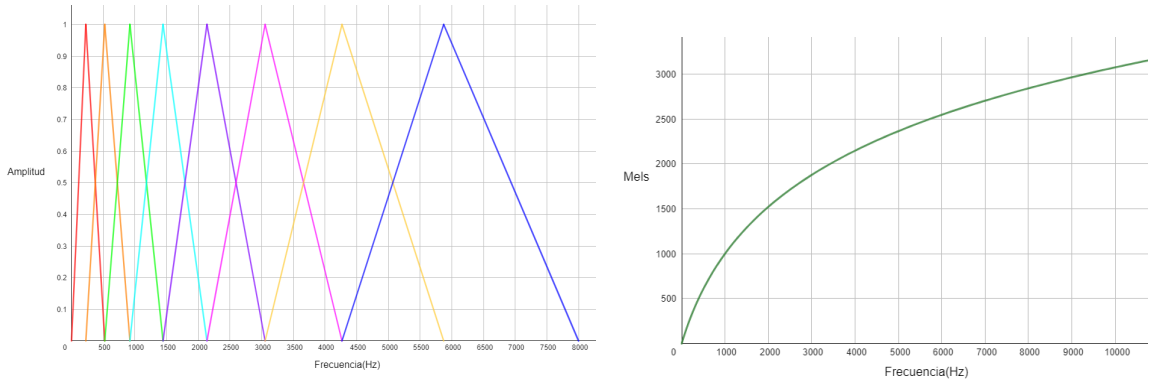

Figura 2.5: Banco de filtros Mel, con 8 filtros y frecuencias entre 0 y 8 kHz (izquierda) y escala de frecuencias Mel (derecha).

Formalmente, los filtros Mel están definidos por:

$$H_m[k] = \begin{cases} 0 & \text{si} \quad k < f[m-1], \\ \frac{2(k-f[m-1])}{(f[m+1]-f[m-1])(f[m]-f[m-1])} & \text{si} \quad f[m-1] \leq k \leq f[m], \\ \frac{2(f[m+1]-k)}{(f[m+1]-f[m-1])(f[m]-f[m-1])} & \text{si} \quad f[m] \leq k \leq f[m+1], \\ 0 & \text{si} \quad k > f[m+1], \end{cases} \tag{2.4}$$

donde $1 \leq m \leq F$, con F representando el número de filtros y la función $f[m]$ determina los extremos de cada banco de frecuencias definidos anteriormente.

Para cada filtro de Mel, se calcula la energía espectral promedio dentro de la banda de frecuencia correspondiente al filtro. Este cálculo se realiza multiplicando los valores de amplitud de la STFT por los coeficientes de los filtros de Mel y sumando los resultados.

2.2.4. Cálculo del logaritmo y normalización

Después de calcular la energía espectral promedio para cada filtro de Mel y obtener los valores correspondientes, se aplica el logaritmo a estos valores. La aplicación del logaritmo tiene dos propósitos principales, comprimir la escala de los valores y ajustarla mejor a las propiedades perceptuales del oído humano. Al aplicar el logaritmo, se reduce la amplitud de los valores más altos y se amplifican los valores más bajos, logrando así una comprensión de la escala. Esto es útil para normalizar la distribución de energía y evitar una sobre-representación de las frecuencias dominantes. Además, la aplicación del logaritmo contribuye a estabilizar la varianza de los valores, lo que puede ser beneficioso para el análisis posterior.

En cuanto a las propiedades perceptuales del oído humano, se ha observado que la percepción de las diferencias en la intensidad de los sonidos es logarítmica en lugar de lineal. Por lo tanto, al aplicar el logaritmo, se ajusta mejor la representación a la manera en que el oído humano percibe las diferencias de energía en el dominio auditivo.

Después de calcular el logaritmo, es común aplicar una etapa de normalización para asegurarse de que los valores estén en un rango adecuado. La normalización puede implicar el ajuste de los valores dentro de un intervalo específico, como $[0, 1]$. Esta etapa de normalización ayuda a mejorar la comparabilidad y la estabilidad de los valores entre diferentes muestras y diferentes características.

2.2.5. Transformada de coseno discreta (DCT)

Por último, para obtener los MFCC se realiza un paso adicional después de calcular los valores logarítmicos de Mel. Este paso implica aplicar la Transformada de Coseno Discreta (DCT), explicada en la sección 1.1,a estos valores logarítmicos.

La DCT es una transformada que se utiliza para extraer las características más relevantes de los coeficientes de energía en el dominio de Mel. Al aplicar la DCT, se reduce la redundancia y la dimensionalidad de los datos, lo que ayuda a obtener una representación más compacta y significativa de la información acústica en forma de los MFCC.

En resumen, el proceso para obtener los MFCC implica el cálculo de los espectros de potencia en la escala de Mel, la aplicación del logaritmo a estos valores y, finalmente, la aplicación de la DCT a los valores resultantes. Este conjunto de pasos permite obtener una representación compacta y significativa de las propiedades acústicas del habla.

Una vez que hemos obtenido los datos de entrada para nuestra red neuronal, en el siguiente capítulo detallaremos minuciosamente el funcionamiento de estas redes y el procesamiento interno de la información.

Capítulo 3

Redes Neuronales para el ASR

Las redes neuronales son modelos computacionales inspirados en el funcionamiento del cerebro humano. Están compuestas por unidades interconectadas, conocidas como neuronas artificiales o nodos, que trabajan en conjunto para procesar información. Estas neuronas se organizan en capas y cada una de ellas realiza operaciones matemáticas en los datos de entrada, generando una salida llamada mapa de características, que se utiliza como entrada para la siguiente capa.

Cada neurona en la red neuronal está conectada con otras neuronas a través de conexiones ponderadas llamadas pesos. Estos pesos determinan la importancia relativa de la información que fluye a través de cada conexión. Durante el proceso de entrenamiento de la red, estos pesos se ajustan gradualmente para que la red pueda aprender a realizar la tarea deseada de manera eficiente. Esto se logra mediante algoritmos de optimización que minimizan una función de pérdida que cuantifica la discrepancia entre las salidas esperadas y las salidas producidas por la red.

Existen diferentes arquitecturas de redes neuronales que se adaptan a diferentes tipos de problemas. Una de ellas es la red neuronal bidireccional convolucional recurrente (BCRNN), que combina elementos de las redes neuronales convolucionales (CNN) y las redes neuronales recurrentes (RNN). Esta arquitectura permite capturar tanto características locales como dependencias a largo plazo en secuencias de datos.

En este capítulo, vamos a explorar en detalle la arquitectura de las BCRNN y su aplicación en tareas de procesamiento de secuencias. Analizaremos cómo estas redes pueden extraer características relevantes de los datos y modelar la dependencia temporal en secuencias.

3.1. Redes Neuronales Bidireccionales Convolucionales Recurrentes

Para abordar la asignación de características extraídas a cada fonema del audio de entrada, una de las técnicas de *deep learning* más utilizadas y con mejores resultados es el uso de modelos que combinan elementos de redes CNN y RNN, como las redes BCRNN.

Como ya hemos mencionado, la combinación de las capas convolucionales y recurrentes en las BCRNN permite capturar tanto las características locales de la señal de entrada como las dependencias a largo plazo entre las diferentes partes de la secuencia de entrada. Las capas convolucionales son eficientes para extraer patrones locales y las capas recurrentes permiten modelar la dependencia temporal en la secuencia de entrada.

Es importante destacar que la arquitectura exacta de una BCRNN puede variar según la aplicación y los requisitos específicos del problema. La configuración de las capas, el número de neuronas y otros hiperparámetros pueden ajustarse de manera personalizada para adaptarse óptimamente a las necesida-

des de cada caso particular. La elección de una arquitectura adecuada es crucial para obtener buenos resultados y maximizar el rendimiento del modelo en la tarea específica que se desea abordar. Por lo tanto, es fundamental realizar un análisis cuidadoso y una experimentación iterativa para determinar la configuración óptima de la BCRNN en cada escenario.

En el contexto de nuestro trabajo, hemos diseñado una configuración específica de capas para la BCRNN, teniendo en cuenta las características de nuestro conjunto de datos y los objetivos de nuestro cometido. La estructura de la red BCRNN empleada en el presente trabajo se muestra en la Figura 3.1.

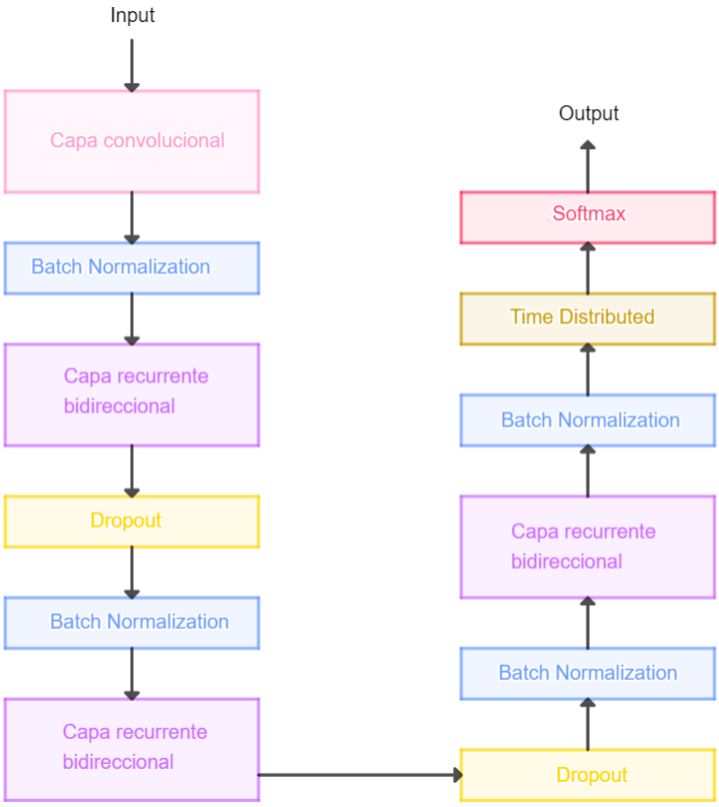

Figura 3.1: Estructura de la red BCRNN empleada en el trabajo.

A continuación, proporcionamos una descripción detallada de las capas utilizadas y su funcionamiento en el procesamiento de las secuencias de datos.

3.1.1. Capa convolucional

En el procesamiento de señales de audio, la capa de convolución se utiliza como una etapa inicial para extraer características locales relevantes, como cambios de frecuencia, duraciones de sonido o patrones de amplitud. Estas características locales se transmiten posteriormente a las capas recurrentes, para capturar dependencias temporales más largas y extraer características más abstractas.

Durante el entrenamiento de la red, la capa de convolución aprende automáticamente los valores de los filtros. Un filtro se define como una matriz de pesos que se aplica deslizándola sobre la señal de entrada, realizando operaciones locales en diferentes regiones. Cada filtro se especializa en detectar un patrón particular en la señal de audio, tales como bordes, formas, texturas u otras características acústicas relevantes para la tarea específica.

A continuación, se explica en detalle la operación de convolución, que es la componente esencial de la capa de convolución.

Operación de convolución

En el ámbito del análisis, la convolución es un operador matemático que se aplica a dos funciones f y g, de variable real, y produce una tercera función denotada como $f * g$. El resultado de esta operación se puede interpretar como el efecto de superponer g con una versión trasladada de f. En una dimensión, la convolución se define de la siguiente manera:

$$(f * g)(x) = \int_{-\infty}^{\infty} f(x-t)g(t)dt. \tag{3.1}$$

Es importante destacar que la definición anterior corresponde a la operación de convolución continua. Sin embargo, en nuestro problema, generalmente los datos de entrada a la red son discretos. Así, podemos definir la operación discreta como sigue:

$$(f * g)(m) = \sum_{n=-\infty}^{\infty} f(m-n)g(n). \tag{3.2}$$

Notar, que si consideramos el caso en el que la entrada a la red es un espectrograma o un MFCC, esta se representa como un vector, por lo que es suficiente con definir la convolución discreta unidimensional, aunque dicha operación exista en n dimensiones.

En el procesamiento de señales se utiliza una operación similar a la convolución llamada correlación cruzada [8]. Usando la misma notación, la correlación cruzada de dos funciones se define como:

$$(f * g)(m) = \sum_{n=-\infty}^{\infty} f(m+n)g(n). \tag{3.3}$$

Es importante destacar que esta definición se asemeja al producto de convolución. De hecho, la relación entre ambas operaciones es la siguiente:

$$(f * g)(m) = \sum_{n=-\infty}^{\infty} f(m+n)g(n) = \sum_{s=-\infty}^{\infty} f(m-s)g(-s) = \sum_{s=-\infty}^{\infty} f(m-s)h(s) = (f * h)(m), \tag{3.4}$$

donde se realiza el cambio $n = -s$ y se define $h(s) = g(-s)$.

En muchas bibliotecas de aprendizaje automático, la operación de correlación cruzada se denomina convolución. En este trabajo, seguiremos este convenio y utilizaremos el término "convolución" para ambas operaciones.

En nuestro modelo, el input de la red consiste en una secuencia de números que puede ser visualizada como un vector. Cada elemento de este vector representa un valor correspondiente a una frecuencia del espectrograma o a un coeficiente de Mel. Durante la operación de convolución, el filtro o *kernel* es aplicado a esta secuencia. El filtro se desliza a lo largo de la secuencia y realiza una multiplicación elemento por elemento con los valores de la secuencia. Luego, se suman los resultados de estas multiplicaciones ponderadas.

El tamaño del filtro en una operación de convolución se determina por la cantidad de elementos que abarca en la dimensión de la secuencia. Por ejemplo, un filtro de tamaño 3 implica que toma tres elementos consecutivos de la secuencia en cada paso. Durante el proceso de convolución, a medida que el filtro se desplaza a lo largo de la secuencia, se realiza la suma de los productos ponderados de los elementos correspondientes. Estos valores resultantes forman la salida de la capa convolucional y se llaman mapas de características. El *stride*, o paso, determina la cantidad de elementos que el filtro se desplaza hacia

la derecha en cada paso durante la operación de convolución. Un *stride* de 1 significa que el filtro se desplaza un elemento cada vez, mientras que un *stride* de 2 implica un desplazamiento de 2 elementos. Un *stride* mayor puede reducir el tamaño de la secuencia resultante y, a su vez, la dimensionalidad de la información.

Gracias a la operación de convolución, la red neuronal puede extraer un nuevo mapa de características del objeto de entrada. En nuestro caso, el nuevo mapa de características sería el vector resultante después del proceso de convolución. Es importante destacar que esta operación también tiene la ventaja de reducir la dimensión del vector de entrada, lo que a su vez reduce el costo computacional en capas posteriores.

Una solución para controlar el tamaño de salida de la convolución y preservar la información espacial del vector de entrada es aplicar una técnica llamada *padding*. El *padding* consiste en agregar ceros en los extremos de la secuencia antes de realizar la convolución.

Además de elegir el valor del *stride* y decidir si aplicar *padding* o no, también podemos seleccionar la matriz filtro que se utilizará en función de las características que deseemos obtener del objeto de entrada. En [18] se pueden encontrar ejemplos de estos tipos de filtros.

En resumen, la convolución desempeña un papel fundamental en la extracción de características relevantes de la secuencia de entrada. Permite identificar patrones y características significativas en dicha secuencia, lo cual es crucial para el rendimiento y la capacidad de aprendizaje de la red neuronal. El resultado de esta operación es un mapa de características formado por neuronas. Cada una de estas neuronas establece conexiones con las neuronas de la capa anterior y de la capa siguiente, es decir, se utiliza como entrada para las capas siguientes, como capas recurrentes o capas completamente conectadas, que procesan la información a un nivel más abstracto y realizan tareas como clasificación o análisis de secuencias.

Cabe destacar que la convolución es una operación lineal, lo que significa que el mapa de características resultante de la convolución también sería lineal. Sin embargo, muchas señales digitales tienen características no lineales, por lo que necesitamos agregar no linealidad a esta capa.

Esto último se puede lograr aplicando una función de activación a cada una de las neuronas obtenidas después del proceso de convolución. Además de agregar la parte no lineal, las funciones de activación permiten decidir si una neurona transmite información a la siguiente capa o no.

Existen diferentes funciones de activación utilizadas en las capas internas. Algunas de las más comunes son ReLU, ELU y la función tangente hiperbólica.

- **Función ReLU:**

$$f(x) = \begin{cases} x & si \quad x > 0, \\ \\ 0 & si \quad x \leq 0. \end{cases} \tag{3.5}$$

- **Función ELU:**

$$f(x) = \begin{cases} x & si \quad x > 0, \\ \\ \alpha(e^x - 1) & si \quad x \leq 0, \end{cases} \tag{3.6}$$

  donde α es un parámetro ajustable.

- **Función de tangente hiperbólica:**

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}. \tag{3.7}$$

Dado que estas funciones de activación desempeñan un papel importante en el aprendizaje de la red, las analizaremos más detalladamente en la sección 3.2.4.

La combinación de la operación de convolución y la aplicación de la función de activación a los elementos de salida forman la capa de convolución. Después de esta capa, es común implementar otra capa para normalizar la salida de la capa de convolución y las capas interconectadas, como capas recurrentes que serán explicadas a continuación.

3.1.2. Capas recurrentes bidireccionales

Como ya se ha mencionado anteriormente, las capas recurrentes bidireccionales son especialmente útiles en el análisis de señales secuenciales, como el habla, debido a su capacidad para capturar dependencias a largo plazo en los datos. Antes de abordar las capas recurrentes bidireccionales, es importante comprender cómo funcionan las capas recurrentes estándar y las capas recurrentes avanzadas, como las LTSM y las GRU.

RNN estándar

La estructura de una red neuronal recurrente (RNN) básica es un conjunto de neuronas recurrentes interconectadas. Cada neurona recurrente está compuesta por una entrada, representada por **x**, una salida, representada por **y** y un estado oculto o *hidden state*, denotado por **h**. En cada paso de tiempo, la neurona recibe dos tipos de entrada, la entrada actual x_t, proveniente de la salida de la neurona anterior, y el estado oculto en el instante de tiempo anterior h_{t-1}, el cual contiene la información de la secuencia de datos procesados hasta el momento, como se muestra en la Figura 3.2 (izquierda).

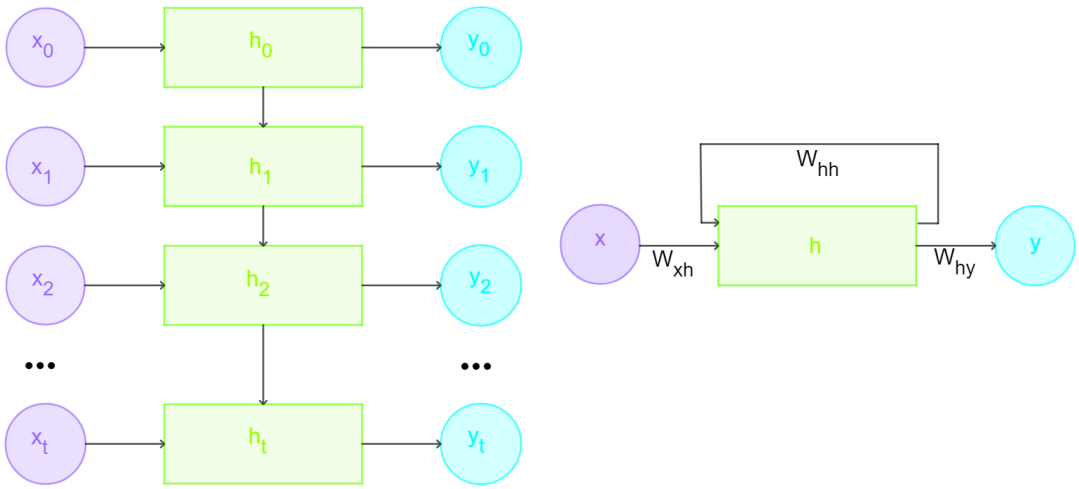

Figura 3.2: Estructura de una red neuronal recurrente básica (izquierda) y de los pesos que intervienen en cada paso (derecha).

En cada neurona recurrente, se utilizan tres conjuntos de pesos: uno para la conexión entre la entrada actual y el estado oculto, que denotamos como W_{xh}, otro para la conexión del estado oculto consigo mismo, denotado como W_{hh}, y otro para la conexión del estado oculto con la salida, W_{hy}, como se muestran en la Figura 3.2 (derecha). Estos pesos se utilizan para calcular la salida actual de la unidad recurrente y también se retroalimentan como entrada en el siguiente paso de tiempo. Esta retroalimentación permite que la RNN mantenga una memoria interna que se actualiza en cada paso de tiempo y captura las dependencias temporales en los datos secuenciales.

En resumen, en una red recurrente básica, el cálculo del estado oculto y la salida se realiza de la siguiente manera:

$$\begin{aligned} h_t &= f_1(W_{xh}\, x_t + W_{hh}\, h_{t-1} + b_h), \\ y_t &= f_2(W_{hy}\, h_t + b_y), \end{aligned} \tag{3.8}$$

donde b_h y b_y son los términos de sesgo que ajustan el punto de partida o nivel de activación de la neurona recurrente, y f_1 y f_2 son funciones de activación que se aplican para incorporar no linealidad a la capa.

Durante el entrenamiento de redes neuronales recurrentes, es común enfrentar desafíos como el desvanecimiento del gradiente y la pérdida de información a medida que se propaga a lo largo del tiempo. Para abordar estos desafíos, se han desarrollado capas recurrentes más complejas que utilizan unidades recurrentes avanzadas, como las LTSM (*Long Short-Term Memory*) y las GRU (*Gated Recurrent Units*).

Long Short-Term Memory (LSTM)

Las LSTM son una variante especializada de las RNN diseñadas para resolver el problema del desvanecimiento del gradiente durante el entrenamiento. El desvanecimiento del gradiente ocurre cuando los gradientes utilizados para actualizar los pesos de la red se vuelven cada vez más pequeños a medida que se propagan hacia atrás en el tiempo, lo que dificulta el aprendizaje de dependencias a largo plazo en secuencias de datos.

Las LSTM son conocidas por su capacidad de incorporar celdas de memoria que permiten capturar y retener información relevante durante largos períodos de tiempo, evitando así la pérdida de información a largo plazo. Esta característica resulta especialmente útil en tareas que involucran secuencias largas o dependencias a largo plazo, ya que les permite comprender y procesar de manera más efectiva la estructura temporal de los datos.

Además de las celdas de memoria, las LSTM también hacen uso de "puertas" para controlar el flujo de información. Estas puertas incluyen la puerta de entrada (*input gate*, i_t), la puerta de olvido (*forget gate*, f_t), y la puerta de salida (*output gate*, o_t). Estas puertas emplean funciones de activación, generalmente sigmoides, $\sigma(x) = \frac{1}{1+e^{-x}}$, para regular la cantidad de información que se agrega, olvida o utiliza en cada paso de tiempo. Cada una de estas puertas genera la salida de una capa de la red, por lo que la arquitectura de una LSTM está compuesta por cuatro capas conectadas.

En cada paso de tiempo, la información que ingresa a cada neurona atraviesa la puerta de olvido, que elimina información no relevante de la iteración anterior. Luego, se agregan nuevas informaciones provenientes de la puerta de entrada, y el resultado se envía directamente fuera de la celda para ser utilizado como entrada en el siguiente instante de tiempo. Por lo tanto, en cada paso se descartan algunos recuerdos mientras se añaden otros nuevos, lo que permite mantener y actualizar la información relevante en la red.

Además, una vez que se agregan nuevos recuerdos, se realiza una copia del estado a largo plazo que luego se filtra mediante la función tangente hiperbólica definida en (3.7) utilizando la puerta de salida para obtener la salida final de la LSTM.

La Figura 3.3 ilustra de dónde provienen los nuevos recuerdos que se memorizan y cómo funcionan estas puertas. Dada una secuencia de entrada actual x_t y un estado oculto en el tiempo anterior h_{t-1}, estas secuencias sirven como información para las cuatro capas conectadas.

La capa principal, que utiliza la función de activación tangente hiperbólica, actúa como una neurona recurrente básica. Analiza las entradas x_t y h_{t-1}, y produce su salida y_t. La diferencia es que la salida de

esta capa no se envía directamente fuera de la neurona, sino que se almacena parcialmente en el estado a largo plazo. Las otras tres capas son controladores de puertas, que utilizan funciones de activación sigmoidea. Esto hace que su salida varíe entre 0 y 1, donde 0 significa que no se transmite ninguna información y 1 significa que se transmite toda la información a la puerta correspondiente.

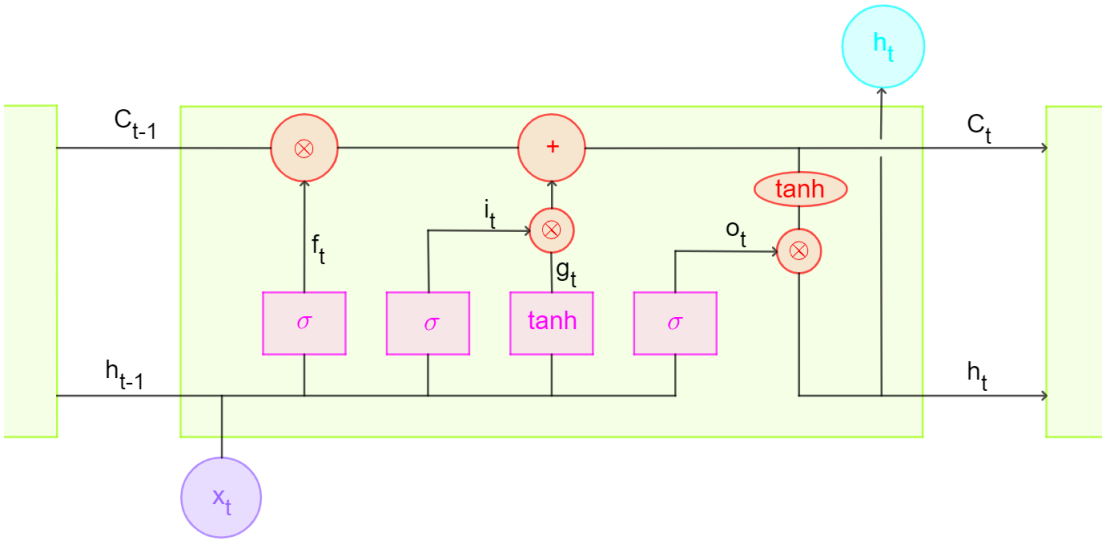

Figura 3.3: Estructura de una neurona LSTM.

En resumen, una neurona LSTM es capaz de capturar, retener y actualizar información relevante a largo plazo, controlando cuidadosamente el flujo de información a través de puertas. Las siguientes ecuaciones resumen cómo se calcula el estado a largo plazo, el estado oculto y la salida para cada paso temporal:

$$\begin{aligned} i_t &= \sigma(W_{xi}\, x_t + W_{hi}\, h_{t-1} + b_i), \\ f_t &= \sigma(W_{xf}\, x_t + W_{hf}\, h_{t-1} + b_f), \\ o_t &= \sigma(W_{xo}\, x_t + W_{ho}\, h_{t-1} + b_o), \\ g_t &= \tanh(W_{xg}\, x_t + W_{hg}\, h_{t-1} + b_g), \\ C_t &= f_t \otimes C_{t-1} + i_t \otimes g_t, \\ y_t &= h_t = o_t \otimes tanh(C_t), \end{aligned} \tag{3.9}$$

donde W_{xi}, W_{xf}, W_{xo} y W_{xg} hacen referencia a las matrices de pesos de las conexiones entre cada una de las capas y la secuencia de entrada, W_{hi}, W_{hf}, W_{ho} y W_{hg} hacen referencia a las matrices de pesos de las conexiones entre cada una de las capas y el estado oculto, b_i, b_f, b_o y b_g son los correspondientes términos de sesgos para cada capa y C_t es el estado de la celda a largo plazo.

Las redes LSTM son eficaces para abordar el problema de las dependencias a largo plazo en el procesamiento de secuencias. Sin embargo, su uso puede ser computacionalmente costoso debido a su estructura más compleja y al mayor número de parámetros. Como resultado, surgieron las *Gated Recurrent Unit* (GRU) como una extensión de las LSTM con un enfoque más ligero y eficiente.

Gated Recurrent Unit (GRU)

Las GRU son una versión simplificada de las LSTM. A diferencia de las LSTM, las GRU no incorporan una celda de memoria, utilizando un enfoque más simple con solo dos puertas: la puerta de

actualización (*update gate*, u_t) y la puerta de reinicio (*reset gate*, r_t), como se muestra en la Figura 3.4.

La puerta de actualización en una GRU decide qué parte de la información pasada se debe mezclar con la información actual, mientras que la puerta de reinicio determina qué parte de la información pasada se debe olvidar. Estas puertas permiten que la GRU controle el flujo de información y mantenga una memoria a corto plazo.

A diferencia de las LSTM, las GRU no tienen una puerta de salida separada. En cambio, la salida de la GRU se calcula directamente utilizando la información actual y la memoria a corto plazo. Esta es la principal simplificación, ya que los vectores de estado a corto y largo plazo se fusionan en un único vector.

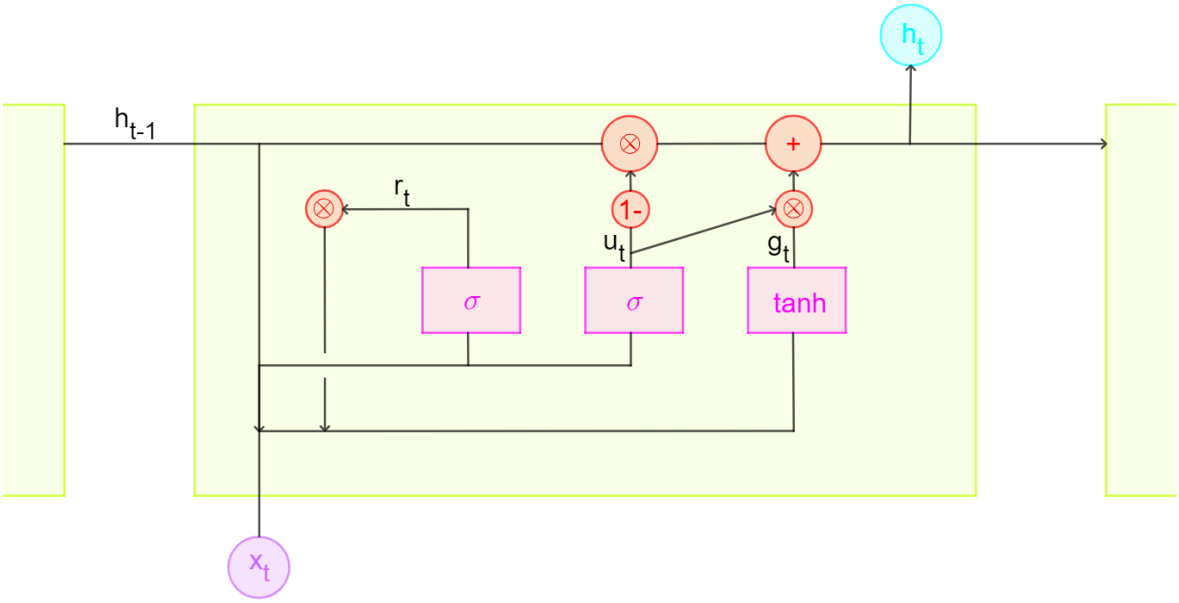

Figura 3.4: Estructura de una neurona GRU.

Las siguientes ecuaciones resumen cómo se calcula la salida para cada paso temporal:

$$\begin{aligned} u_t &= \sigma(W_{xu}\, x_t + W_{hu}\, h_{t-1} + b_u), \\ r_t &= \sigma(W_{xr}\, x_t + W_{hr}\, h_{t-1} + b_r), \\ g_t &= tanh(W_{xg}\, x_t + W_{hg}\, (r_t \otimes h_{t-1} + b_g), \\ h_t &= (1-u_t) \otimes h_{t-1} + u_t \otimes g_t, \end{aligned} \tag{3.10}$$

donde W_{xu}, W_{xg} y W_{xr} hacen referencia a las matrices de pesos de las conexiones entre cada una de las capas y la secuencia de entrada, W_{hu}, W_{hg} y W_{hr} hacen referencia a las matrices de pesos de las conexiones entre cada una de las capas y el estado oculto y b_u, b_r y b_g son los correspondientes términos de sesgos para cada capa.

En resumen, las neuronas GRU son una alternativa más ligera y eficiente a las LSTM, ya que simplifican la estructura y reducen el costo computacional. Aunque las LSTM siguen siendo relevantes y efectivas en muchas aplicaciones, las GRU son una opción a considerar cuando se busca un equilibrio entre el rendimiento y la eficiencia en el procesamiento de secuencias.

Este tipo de neuronas resuelve el problema del desvanecimiento del gradiente. Sin embargo, aún persisten desafíos relacionados con la unidireccionalidad de la red, como la limitación para capturar dependencias de largo alcance en la secuencia de entrada. Estos desafíos pueden abordarse mediante el uso de redes recurrentes bidireccionales.

RNN bidireccional

La arquitectura de una red neuronal recurrente bidireccional es una extensión de la RNN estándar que permite capturar información tanto del pasado como del futuro de una secuencia durante el proceso de aprendizaje. Esto se logra mediante la adición de otra capa oculta que recorre la secuencia en sentido inverso.

En la RNN bidireccional, se utilizan dos conjuntos de unidades recurrentes: una secuencia que procesa los datos de forma secuencial en orden ascendente (de pasado a futuro) y otra secuencia que procesa los datos en orden descendente (de futuro a pasado). Cada conjunto de unidades recurrentes tiene sus propios pesos y conexiones recurrentes.

La arquitectura de la RNN bidireccional se puede visualizar como la combinación de estos dos conjuntos, como se muestra en la Figura 3.5. En esta arquitectura, las salidas de las unidades recurrentes en cada conjunto se combinan para producir la salida final de la RNN bidireccional.

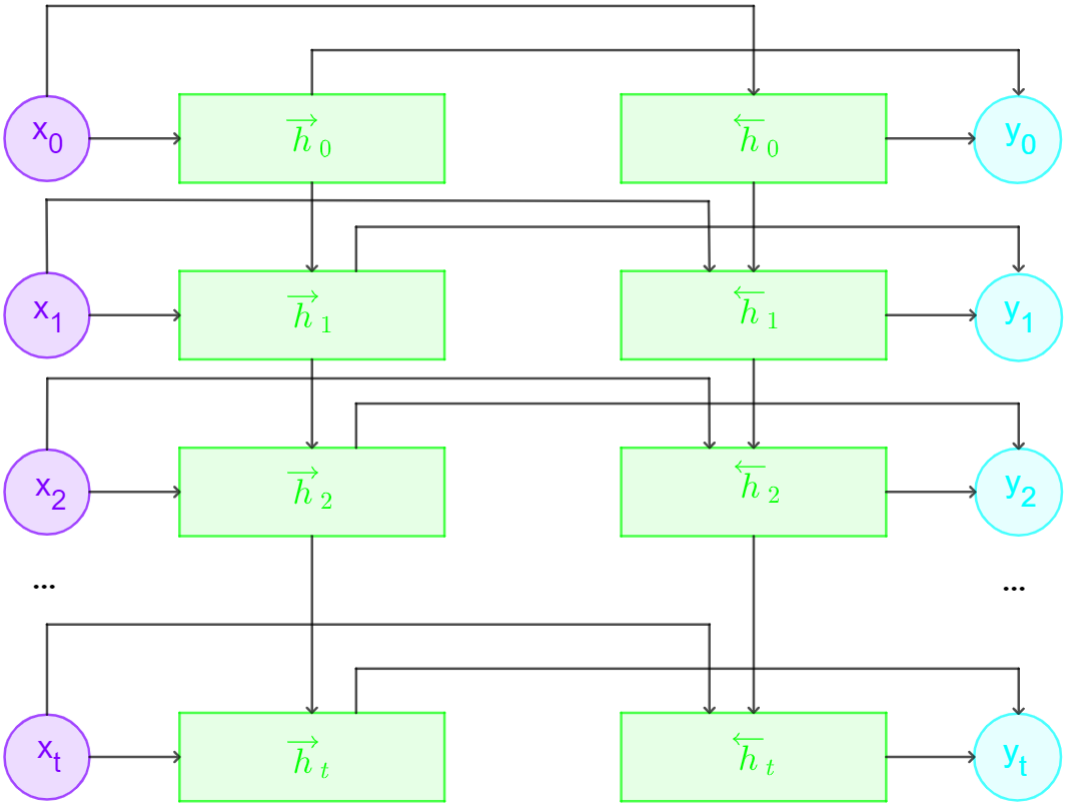

Figura 3.5: Arquitectura de una red neuronal recurrente bidireccional.

La salida de una neurona recurrente en este contexto se puede calcular de la siguiente manera:

$$\begin{aligned} \overrightarrow{h}_t &= f_1(W_{xh}\, x_t + W_{hh}\, \overrightarrow{h}_{t-1} + b_h), \\ \overleftarrow{h}_t &= f_1(W_{xh}\, x_t + W_{hh}\, \overleftarrow{h}_{t-1} + b_h), \\ y_t &= f_2([\overrightarrow{h}_t, \overleftarrow{h}_t]\, W_{hy} + b_y), \end{aligned} \tag{3.11}$$

donde W_{xh}, W_{hh} y W_{hy} son las matrices de pesos correspondientes, b_h y b_y son los términos de sesgos y $[\overrightarrow{h}(t), \overleftarrow{h}(t)]$ denota la concatenación de las dos matrices de estados ocultos.

Es posible modificar esta estructura de RNN bidireccional, reemplazando las unidades recurrentes básicas por LSTM o GRU, que como hemos visto, son versiones más complejas y capaces de capturar dependencias a largo plazo en las secuencias. Estas variantes ofrecen un mayor poder de modelado y han demostrado ser eficaces en una amplia gama de tareas de procesamiento de secuencias, como el reconocimiento del habla [24, 27].

En nuestro trabajo, hemos optado por utilizar una estructura de red neuronal recurrente (RNN) bidireccional con unidades GRU en lugar de LSTM. La elección se basa en que las GRU tienen un menor coste computacional en comparación con las LSTM, lo cual nos permite obtener resultados eficientes sin comprometer el rendimiento del modelo.

Al igual que ocurre con la capa convolucional, después de aplicar la capa recurrente en un modelo, es común utilizar una capa para normalizar la salida de la capa y mejorar la estabilidad y el rendimiento del modelo.

3.2. Aprendizaje de la red

Una vez presentados todos los elementos de una BCRNN completa y las entradas de dicha red, se procede a explicar el proceso de aprendizaje de la red. El aprendizaje de una red se basa en ajustar los pesos de la misma para lograr la precisión deseada. Para lograr esto, se utiliza un conjunto de patrones de muestra o entrenamiento, que en este caso son secuencias de datos obtenidas a partir de los audios, junto con sus respectivas transcripciones textuales.

Estos patrones de entrenamiento representan las secuencias de audio que se utilizan para enseñar a la red cómo reconocer y clasificar diferentes patrones en el sonido. Cada patrón de entrenamiento consiste en una secuencia de valores numéricos que representan las amplitudes del sonido a lo largo del tiempo.

Durante el entrenamiento, la red ejecuta iterativamente los patrones de entrenamiento, ajustando los parámetros hasta converger hacia un conjunto óptimo de pesos que representen con precisión dichos patrones. En otras palabras, los parámetros se ajustan para proporcionar respuestas correctas a los patrones de entrenamiento presentados.

Antes de comenzar el entrenamiento de nuestra red, es conveniente separar nuestro conjunto de datos en dos subconjuntos llamados conjunto de entrenamiento y conjunto de test o validación. El objetivo de esta división es el de poder evaluar un posible sobreajuste, donde la red aprende demasiado bien los patrones de entrenamiento específicos, pero tiene un rendimiento deficiente en datos nuevos. Trabajar con estos dos conjuntos nos permite entrenar la red con el conjunto de entrenamiento y luego evaluar su precisión utilizando el conjunto de validación. De esta manera, podemos ajustar la arquitectura de la red y seleccionar aquella que proporcione los mejores resultados.

En el proceso de aprendizaje de una red neuronal, las funciones de pérdida juegan un papel fundamental. Estas funciones cuantifican la discrepancia o diferencia entre las salidas predichas por la red y las salidas esperadas, proporcionando una medida del rendimiento del modelo. El objetivo del entrenamiento es minimizar esta función de pérdida, lo que implica ajustar los parámetros de la red para que las predicciones se acerquen lo más posible a los valores deseados. Al minimizar la función de pérdida, la red aprende a realizar la tarea específica para la cual ha sido diseñada.

3.2.1. Función de perdida

Existen diversas funciones de pérdida que se utilizan en función del tipo de problema y del modelo empleado. En nuestro modelo, en el que utilizamos retroalimentación, nos enfrentamos a desafíos que

dificultan el uso de algoritmos de aprendizaje simples. Estos desafíos incluyen la variabilidad en la longitud de las secuencias de entrada y salida, la falta de alineación precisa entre ellas y la variación en las proporciones de longitud entre las secuencias.

Para abordar estos desafíos, se ha desarrollado la clasificación temporal de conexiones (CTC), un algoritmo creado en 1990 por Alex Graves [9]. El CTC utiliza las probabilidades para calcular la diferencia entre la secuencia de salida predicha por la red y la secuencia de transcripción real asociada al audio.

El objetivo principal de la función de pérdida en el contexto de CTC es entrenar nuestro modelo para maximizar la probabilidad asignada a la respuesta correcta. Para lograr esto, necesitamos calcular de manera eficiente la probabilidad condicional de las alineaciones entre la secuencia de entrada y salida, teniendo en cuenta la flexibilidad requerida por el problema.

Durante el proceso de aprendizaje de la red, utilizaremos el conjunto de entrenamiento $S = \{s_1, ..., s_k\}$, donde k es el número de observaciones que tendremos en cuenta. Este conjunto está formado por pares de elementos $s = (\mathbf{x}, \mathbf{z})$, donde $\mathbf{x}$ representa la secuencias de entrada de la red con longitud T, en nuestro caso, los MFCC o los valores de los espectrogramas, y $\mathbf{z}$ representa los valores reales de las etiquetas, es decir, las transcripciones reales de las secuencias de audio.

Recordar que el objetivo principal de la función de pérdida CTC es maximizar la probabilidad de la secuencia correcta durante el entrenamiento. Esto se logra minimizando simultáneamente el negativo del logaritmo de las probabilidades de todas las alineaciones correctas en el conjunto de entrenamiento. Por lo tanto, la función de perdida CTC se define en cada instante de tiempo t de la siguiente manera:

$$\mathscr{L}_t(\theta; S) = -\sum_{(\mathbf{x},\mathbf{z})\in S} \ln P_\theta(\mathbf{z}|\mathbf{x}), \tag{3.12}$$

donde θ es el vector de pesos que permite calcular las salidas de la red a partir de una función, $\mathbf{y} = \mathscr{N}_\theta(\mathbf{x})$.

El objetivo ahora es encontrar una manera de calcular $P_\theta(\mathbf{z}|\mathbf{x})$. Para ello, definimos el conjunto L de clases o etiquetas, que en nuestro caso incluye todas las letras del alfabeto castellano junto con el espacio en blanco, denotado como ε, y las vocales con tilde. Además, denotamos por y_t^r a la activación de la unidad de salida r en el instante de tiempo t, lo cual se puede interpretar como la probabilidad de observar la etiqueta r en el instante t. Estas salidas están restringidas al rango $[0, 1]$, gracias a la utilización de funciones de activación adecuadas, lo que permite definir una distribución sobre el conjunto L^T, que representa todas las posibles alineaciones de longitud T. Esta distribución viene dada de la siguiente manera:

$$P_\theta(\pi|\mathbf{x}) = \prod_{t=0}^{T} y_t^{\pi_t},\ \forall \pi = (\pi_1, ..., \pi_T) \in L^T, \tag{3.13}$$

donde π_t es la etiqueta correspondiente a la alineación π en el instante de tiempo t.

Con esto, podemos escribir la probabilidad condicional de $\mathbf{z}$ dado $\mathbf{x}$ en función de las probabilidades (3.13) como:

$$P_\theta(\mathbf{z}|\mathbf{x}) = \sum_{\pi\in\mathscr{B}^{-1}(\mathbf{z})} P_\theta(\pi|\mathbf{x}), \tag{3.14}$$

donde $\mathscr{B}^{-1}$ es la función inversa de aquella que asigna a las posibles alineaciones de longitud T, alineaciones de longitud menor o igual que T, generadas al eliminar los espacios en blanco y las etiquetas repetidas.

Enumerar y calcular directamente todas las posibles alineaciones de longitud T puede ser computacionalmente costoso y poco práctico. Por lo tanto, para calcular de manera eficiente $P_\theta(\pi|\mathbf{x})$ se implementa el algoritmo *CTC Forward-Backward*. La idea principal detrás de este algoritmo es descomponer

la suma sobre las trayectorias correspondientes a una etiqueta en una suma iterativa sobre las trayectorias correspondientes a prefijos de esa etiqueta. Las iteraciones se pueden calcular de manera eficiente utilizando probabilidades recursivas de avance y retroceso. Consideramos un ejemplo para entender mejor el algoritmo.

Ejemplo 1 (Secuencia $\mathbf{z}$ = "*perro*")**.** Supongamos que tenemos una secuencia de entrada $\mathbf{x}$, con su etiquetado $\mathbf{z}$ = "*perro*". Para permitir la presencia de elementos en blanco en las trayectorias de salida, se considera una nueva secuencia $\mathbf{z}'$, con espacios en blanco agregados al principio y al final e insertados entre cada par de etiquetas. Entonces, $\mathbf{z}'$ = "$\varepsilon p \varepsilon e \varepsilon r \varepsilon r \varepsilon o \varepsilon$". Recordar que la secuencia de entrada $\mathbf{x}$ tiene longitud T, que puede variar según las características obtenidas de la señal de audio. Por simplicidad, en este ejemplo, asumiremos que $T = 7$. □

Para facilitar la comprensión del algoritmo CTC *Forward-Backward*, combinamos este ejemplo con la explicación teórica del cálculo de las probabilidades que nos permitirán llevar a cabo el algoritmo.

En primer lugar, nos enfocamos en el cálculo de las probabilidades *forward* $\alpha_{s,t}$, que representan la probabilidad acumulada de todas las posibles alineaciones que asignan la etiqueta s en el instante t. Existen dos posibles casos para calcular estas probabilidades:

- Si no se puede omitir la etiqueta anterior z'_{s-1}, esto ocurre en dos casos: cuando $z'_s = \varepsilon$ (indicando que la etiqueta anterior es un elemento de la secuencia de salida) o cuando $z'_{s-2} = z'_s$ (indicando que se necesita introducir un símbolo en blanco entre etiquetas repetidas de la secuencia de salida). En ambos casos, hay un elemento repetido en la secuencia que no se puede eliminar. En este caso, la probabilidad $\alpha_{s,t}$ depende únicamente de la etiqueta anterior y se calcula de la siguiente manera:

$$\alpha_{s,t} = (\alpha_{s-1,t-1} + \alpha_{s,t-1}) \cdot y_t^{z'_s}. \tag{3.15}$$

Ejemplo 2 (Secuencia $\mathbf{z}$ = "*perro*" (cont.))**.** Consideramos $s = 3$ y $t = t_2$. Recordar que $\mathbf{z}'$ = "$\varepsilon p \varepsilon e \varepsilon r \varepsilon r \varepsilon o \varepsilon$", por lo tanto $z'_3 = \varepsilon$. Para calcular α_{3,t_2} necesitamos calcular la probabilidad acumulada de todas las posibles alineaciones que asignan la etiqueta ε en el instante $t = t_2$.

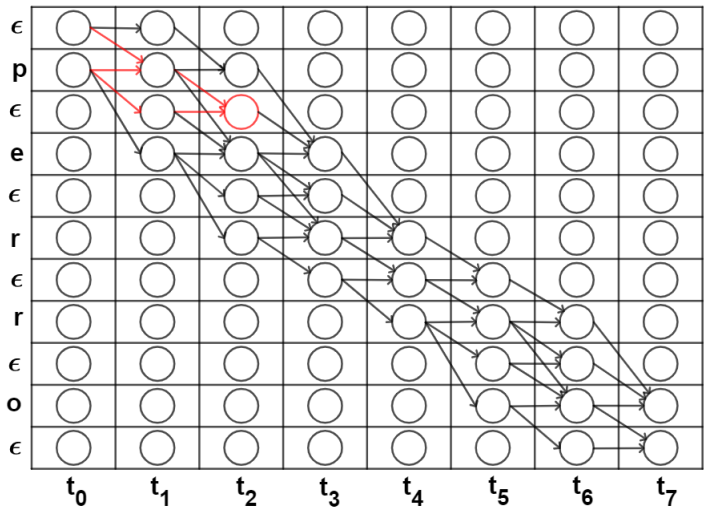

Figura 3.6: Ejemplo del cálculo de la probabilidad *forward* α_{3,t_2} en el etiquetado $\mathbf{z}$ = "*perro*". Las flechas negras indican todas las posibles alineaciones para completar el etiquetado. Y las flechas rojas indican las posibles alineaciones que asignan la etiqueta ε en el instante $t = t_2$.

Observando las flechas rojas en la Figura 3.6, identificamos las siguientes posibles alineaciones: "$\varepsilon p \varepsilon$", "$pp\varepsilon$" y "$p\varepsilon\varepsilon$" . Estas probabilidades se pueden calcular usando la ecuación (3.13). Así,

obtenemos:

$$\begin{aligned}\alpha_{3,t_2} = P_\theta(\text{“}\varepsilon p\varepsilon\text{”}|\mathbf{x}) + P_\theta(\text{“}pp\varepsilon\text{”}|\mathbf{x}) + P_\theta(\text{“}p\varepsilon\varepsilon\text{”}|\mathbf{x}) = \\ y^{\varepsilon}_{t_0}\, y^{p}_{t_1}\, y^{\varepsilon}_{t_2} + y^{p}_{t_0}\, y^{p}_{t_1}\, y^{\varepsilon}_{t_2} + y^{p}_{t_0}\, y^{\varepsilon}_{t_1}\, y^{\varepsilon}_{t_2} = (\alpha_{2,t_1} + \alpha_{3,t_1}) y^{\varepsilon}_{t_2}.\end{aligned} \tag{3.16}$$

□

- Si se puede omitir z'_{s-1}, es decir, cuando es un símbolo en blanco entre etiquetas distintas. En este caso, la probabilidad $\alpha_{s,t}$ depende de las dos etiquetas anteriores y se calcula de la siguiente manera:

$$\alpha_{s,t} = (\alpha_{s-2,t-1} + \alpha_{s-1,t-1} + \alpha_{s,t-1}) \cdot y_t^{z'_s}. \tag{3.17}$$

Ejemplo 3 (Secuencia **z**=“perro”(cont.))**.** Consideramos en el ejemplo $s = 4$ y $t = t_2$. Recordemos que $z'_4 = e$. Procedemos de manera similar para calcular α_{4,t_2}. Observando las flechas moradas en la Figura 3.7, identificamos las siguientes posibles alineaciones: “εpe”, “ppe”, “$p\varepsilon e$” y “pee”.

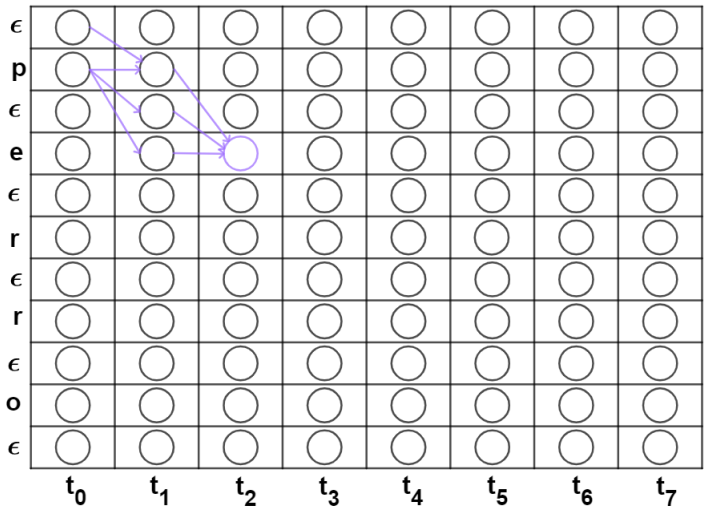

Figura 3.7: Ejemplo del cálculo de la probabilidad *forward* α_{4,t_2} en el etiquetado $\mathbf{z} =$ “*perro*”.

Por lo tanto, la probabilidad viene dada por:

$$\begin{aligned}\alpha_{4,t_2} = P_\theta(\text{“}\varepsilon pe\text{”}|\mathbf{x}) + P_\theta(\text{“}ppe\text{”}|\mathbf{x}) + P_\theta(\text{“}p\varepsilon e\text{”}|\mathbf{x}) + P_\theta(\text{“}pee\text{”}|\mathbf{x}) = \\ y^{\varepsilon}_{t_0}\, y^{p}_{t_1}\, y^{e}_{t_2} + y^{p}_{t_0}\, y^{p}_{t_1}\, y^{e}_{t_2} + y^{p}_{t_0}\, y^{\varepsilon}_{t_1}\, y^{e}_{t_2} + y^{p}_{t_0}\, y^{e}_{t_1}\, y^{e}_{t_2} = (\alpha_{2,t_1} + \alpha_{3,t_1} + \alpha_{4,t_1}) y^{e}_{t_2}.\end{aligned} \tag{3.18}$$

□

Estos casos permiten calcular de forma recursiva las probabilidades *forward* como:

$$\alpha_{s,t} = \begin{cases} (\alpha_{s-1,t-1} + \alpha_{s,t-1}) \cdot y_t^{z'_s} & \text{si } z'_s = \varepsilon \text{ o } z'_{s-2} = z'_s, \\ (\alpha_{s-2,t-1} + \alpha_{s-1,t-1} + \alpha_{s,t-1}) \cdot y_t^{z'_s} & \text{en otro caso.} \end{cases} \tag{3.19}$$

Análogamente, definimos las probabilidades *backward* $\beta_{s,t}$, que representan la probabilidad acumulada de todas las alineaciones que asignan los últimos símbolos del etiquetado desde el tiempo t hasta el final de la secuencia. Al igual en las probabilidades *forward*, el cálculo se realiza de manera recursiva de la siguiente manera:

$$\beta_{s,t} = \begin{cases} (\beta_{s+1,t+1} + \beta_{s,t+1}) \cdot y_t^{z'_s} & \text{si } z'_s = \varepsilon \text{ o } z'_{s+2} = z'_s, \\ (\beta_{s+2,t+1} + \beta_{s+1,t+1} + \beta_{s,t+1}) \cdot y_t^{z'_s} & \text{en otro caso.} \end{cases} \tag{3.20}$$

Ejemplo 4 (Secuencia **z**="perro" (cont.))**.** Finalmente, veamos cómo se combinan ambas probabilidades para obtener la probabilidad de que en el instante $t = t_2$ la etiqueta sea p. Observando las flechas azules en la Figura 3.8, se tiene:

$$\alpha_{2,t_2} = P_\theta(\text{“}\varepsilon\varepsilon p\text{”}|\mathbf{x}) + P_\theta(\text{“}\varepsilon pp\text{”}|\mathbf{x}) + P_\theta(\text{“}ppp\text{”}|\mathbf{x}) = y^{\varepsilon}_{t_0}\, y^{\varepsilon}_{t_1}\, y^{p}_{t_2} + y^{\varepsilon}_{t_0}\, y^{p}_{t_1}\, y^{p}_{t_2} + y^{p}_{t_0} y^{p}_{t_1}\, y^{p}_{t_2}. \tag{3.21}$$

La única posibilidad para completar el etiquetado, como indican las flechas rosas en la Figura 3.8, es la alineación "$per\varepsilon ro$", por lo tanto, la probabilidad β_{2,t_2} se calcula como:

$$\beta_{2,t_2} = P_\theta(\text{“}per\varepsilon ro\text{”}|\mathbf{x}) = y^{p}_{t_2}\, y^{e}_{t_3}\, y^{r}_{t_4}\, y^{\varepsilon}_{t_5}\, y^{r}_{t_6}\, y^{0}_{t_7} = y^{p}_{t_2}\, \beta_{4,t_3}, \tag{3.22}$$

ya que $\beta_{3,t_3} = \beta_{2,t_3} = 0$.

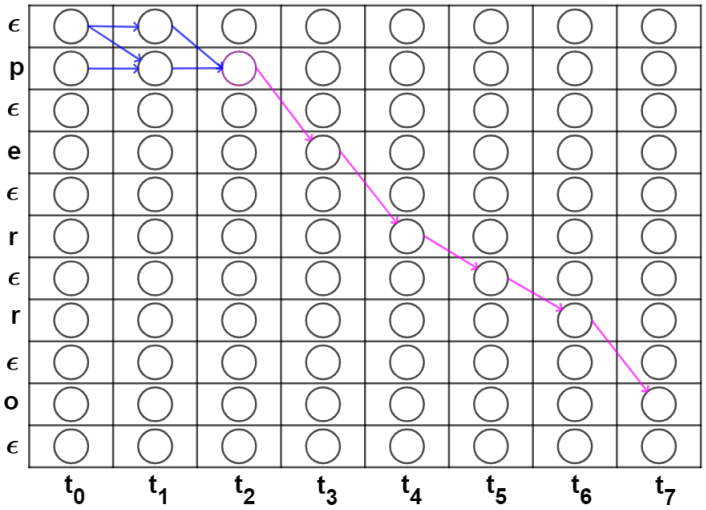

Figura 3.8: Ejemplo del algoritmo CTC *Forward-Backward* en la etiqueta $\mathbf{z} =$ "*perro*".

Por lo tanto,

$$\alpha_{2,t_2} \cdot \beta_{2,t_2} = y^{\varepsilon}_{t_0}\, y^{\varepsilon}_{t_1}\, y^{p}_{t_2}\, y^{p}_{t_2}\, y^{e}_{t_3}\, y^{r}_{t_4}\, y^{\varepsilon}_{t_5}\, y^{r}_{t_6}\, y^{0}_{t_7} + y^{\varepsilon}_{t_0}\, y^{p}_{t_1}\, y^{p}_{t_2}\, y^{p}_{t_2}\, y^{e}_{t_3}\, y^{r}_{t_4}\, y^{\varepsilon}_{t_5}\, y^{r}_{t_6}\, y^{0}_{t_7} + y^{p}_{t_0} y^{p}_{t_1}\, y^{p}_{t_2}\, y^{p}_{t_2}\, y^{e}_{t_3}\, y^{r}_{t_4}\, y^{\varepsilon}_{t_5}\, y^{r}_{t_6}\, y^{0}_{t_7} =$$

$$= [P_\theta(\text{“}\varepsilon\varepsilon per\varepsilon ro\text{”}|\mathbf{x}) + P_\theta(\text{“}\varepsilon pper\varepsilon ro\text{”}|\mathbf{x}) + P_\theta(\text{“}ppper\varepsilon ro\text{”}|\mathbf{x})]\, y^{p}_{t_2}. \tag{3.23}$$

Es decir, la probabilidad total de todas las alineaciones que pasan por p en el instante $t = t_2$ es:

$$\frac{\alpha_{2,t_2} \cdot \beta_{2,t_2}}{y^{p}_{t_2}} \tag{3.24}$$

En general, la probabilidad del etiquetado $\mathbf{z} =$ "*perro*" en el instante t se puede expresar como:

$$P(\text{“}perro\text{”}|\mathbf{x}) = \sum_{s=1}^{|z'|=11} \frac{\alpha_{s,t} \cdot \beta_{s,t}}{y^{z'_s}_{t}}. \tag{3.25}$$

□

Una vez hemos revisado el ejemplo, deducimos las fórmulas de forma general. El punto clave radica en que, para un etiquetado $\mathbf{z}$, el producto de las probabilidades *forward* y *backward* en un determinado s en el instante de tiempo t representa la probabilidad de todas las alineaciones que corresponden a $\mathbf{z}$ y atraviesan el símbolo s en el instante t. Más precisamente,

$$\frac{\alpha_{s,t} \cdot \beta_{s,t}}{y^{z'_s}_{t}} = \sum_{\pi \in \mathscr{B}^{-1}(\pi)} P_\theta(\pi|\mathbf{x}). \tag{3.26}$$

A partir de la ecuación (3.14), podemos ver que esto es la porción de la probabilidad total $P_\theta(\mathbf{z}|\mathbf{x})$ debida a las alineaciones que pasan por z'_s en el instante t. Luego para cada t, podemos sumar sobre todos los valores de s para obtener:

$$P_\theta(\mathbf{z}|\mathbf{x}) = \sum_{s=1}^{|z'|} \frac{\alpha_{s,t} \cdot \beta_{s,t}}{y_t^{z'_s}}. \tag{3.27}$$

Finalmente, se puede expresar la función de pérdida CTC en cada instante de tiempo t en términos de la probabilidad (3.27) como:

$$\mathscr{L}_t(\theta) = -\sum_{(\mathbf{x},\mathbf{z})\in S} ln(P_\theta(\mathbf{z}|\mathbf{x})) = -\sum_{(\mathbf{x},\mathbf{z})\in S} ln\left(\sum_{s=1}^{|z'|} \frac{\alpha_{s,t}\beta_{s,t}}{y^t_{z'_s}}\right). \tag{3.28}$$

Una vez que hemos calculado la función de pérdida, el siguiente paso es utilizar el algoritmo de retropropagación a través del tiempo (BPTT) para ajustar los pesos de la red y minimizar dicha función.

3.2.2. Algoritmo de retropropagación a través del tiempo (BPTT)

El BPTT es una adaptación del algoritmo de retropropagación estándar utilizado para entrenar redes neuronales recurrentes (RNN) [30]. En esencia, desenrolla la RNN en el tiempo, creando una red neuronal desplegada similar a una red *feedforward* convencional. Esto permite aplicar el algoritmo de retropropagación estándar para calcular los gradientes y actualizar los pesos de la red.

Notar, que podemos expresar cualquier función de pérdida como una suma en cada instante de tiempo t de la siguiente manera:

$$\mathscr{L}(\theta) = \sum_{t=0}^{T} l_t(\theta), \tag{3.29}$$

donde θ es el vector de pesos que permite calcular las salidas de la red a partir de una función.

Vamos a analizar el BPTT en el caso específico de una RNN estándar. Para un análisis más detallado con respecto a redes bidireccionales se puede consultar [5]. Recordemos que en este contexto, tenemos tres matrices de pesos, $\theta = \{W_{xh}, W_{hh}, W_{hy}\}$, y cada salida de la neurona se calcula de la siguiente manera:

$$\begin{aligned} h_t &= f_1(W_{xh}\, x_t + W_{hh}\, h_{t-1} + b_h), \\ y_t &= f_2(W_{hy}\, h_t + b_y). \end{aligned} \tag{3.30}$$

Para ajustar estos pesos, se deben calcular las derivadas parciales de la función de pérdida con respecto a cada una de estas matrices. Esto se realiza utilizando la regla de la cadena. Siguiendo el enfoque propuesto en el artículo [23] adaptado a nuestro modelo y asumiendo que la función de activación f_2 es una *softmax*, que se introducirá más adelante en (3.45), las derivadas parciales se calculan de la siguiente manera:

$$\begin{aligned} \frac{\partial \mathscr{L}}{\partial W_{hy}} &= \sum_{t=0}^{T} \frac{\partial l_t}{\partial y_t} \cdot \frac{\partial y_t}{\partial f_2} \cdot \frac{\partial f_2}{\partial W_{hy}} = \sum_{t=0}^{T} \frac{\partial l_t}{\partial y_t} \cdot \frac{\partial y_t}{\partial f_2} \cdot h_t, \\ \frac{\partial \mathscr{L}}{\partial W_{hh}} &= \sum_{t=0}^{T} \frac{\partial l_t}{\partial y_t} \cdot \frac{\partial y_t}{\partial f_2} \cdot \frac{\partial f_2}{\partial h_t} \cdot \frac{\partial h_t}{\partial f_1} \cdot \frac{\partial f_1}{\partial W_{hh}} = \sum_{t=0}^{T} \frac{\partial l_t}{\partial y_t} \cdot \frac{\partial y_t}{\partial f_2} \cdot W_{hy} \cdot \frac{\partial h_t}{\partial f_1} \cdot \frac{\partial f_1}{\partial W_{hh}}, \\ \frac{\partial \mathscr{L}}{\partial W_{xh}} &= \sum_{t=0}^{T} \frac{\partial l_t}{\partial y_t} \cdot \frac{\partial y_t}{\partial f_2} \cdot \frac{\partial f_2}{\partial h_t} \cdot \frac{\partial h_t}{\partial f_1} \cdot \frac{\partial f_1}{\partial W_{xh}} = \sum_{t=0}^{T} \frac{\partial l_t}{\partial y_t} \cdot \frac{\partial y_t}{\partial f_2} \cdot W_{hy} \cdot \frac{\partial h_t}{\partial f_1} \cdot \frac{\partial f_1}{\partial W_{xh}}. \end{aligned} \tag{3.31}$$

Dado que el estado oculto depende del instante de tiempo anterior, las dos últimas derivadas parciales

se pueden reescribir como:

$$\begin{aligned}\frac{\partial \mathscr{L}}{\partial W_{hh}} &= \sum_{t=0}^{T} \frac{\partial l_t}{\partial y_t} \cdot \frac{\partial y_t}{\partial f_2} \cdot W_{hy} \sum_{k=0}^{t} \frac{\partial h_t}{\partial h_k} \cdot \frac{\partial h_k}{\partial W_{hh}}, \\ \frac{\partial \mathscr{L}}{\partial W_{xh}} &= \sum_{k=0}^{t} \frac{\partial l_t}{\partial y_t} \cdot \frac{\partial y_t}{\partial f_2} \cdot W_{hy} \sum_{k=0}^{t} \frac{\partial h_t}{\partial h_k} \cdot \frac{\partial h_k}{\partial W_{xh}}.\end{aligned} \tag{3.32}$$

Es importante destacar que, para que estas derivadas parciales existan, es necesario que exista $\frac{\partial l_t}{\partial y_t}$. Vamos a analizar qué ocurre en el caso particular de la función de pérdida CTC. Recordemos que la función de pérdida CTC dada por la ecuación (3.28) representa la función de pérdida en cada instante de tiempo. Por lo tanto, se puede expresar de la forma (3.29), como:

$$\mathscr{L}(\theta) = \sum_{t=0}^{T} \mathscr{L}_t(\theta). \tag{3.33}$$

Por lo tanto, en este caso particular, debemos asegurarnos de que exista:

$$\frac{\partial \mathscr{L}_t}{\partial y_t}. \tag{3.34}$$

Dado que los ejemplos de entrenamiento son independientes, podemos considerarlos por separado, y calcular la derivada con respecto a una salida específica de la red:

$$\frac{\partial \mathscr{L}_t}{\partial y_t^r} = -\frac{\partial ln(P_\theta(\mathbf{z}|\mathbf{x}))}{\partial y_t^r}. \tag{3.35}$$

Utilizando (3.27), llegamos a:

$$\frac{\partial P_\theta(\mathbf{z}|\mathbf{x})}{\partial y_t^r} = \frac{1}{y_t^{r2}} \sum_{s \in lab(\mathbf{z},r)} \alpha_{s,t} \cdot \beta_{s,t}, \tag{3.36}$$

donde $lab(\mathbf{z},r) = \{s : z'_s = r\}$, es decir, es el conjunto de posiciones donde aparece la etiqueta r en la secuencia $\mathbf{z}$.

Finalmente, utilizando la regla de la cadena, tenemos:

$$\frac{\partial (lnP_\theta(\mathbf{z}|\mathbf{x}))}{\partial y_t^r} = \frac{1}{P_\theta(\mathbf{z}|\mathbf{x})} \frac{\partial (P_\theta(\mathbf{z}|\mathbf{x}))}{\partial y_t^r} = y_t^r - \frac{1}{y_t^r Z_t} \sum_{s \in lab(\mathbf{z},r)} \alpha_{s,t} \cdot \beta_{s,t}, \tag{3.37}$$

donde $Z_t = \sum_{s=1}^{|z'|} \frac{\alpha_{s,t} \cdot \beta_{s,t}}{y_t^{z'_s}}$.

Una vez obtenidas las derivadas parciales, se utilizan en el proceso de actualización de pesos, donde se minimiza el error de predicción de la red neuronal.

Sin embargo, es importante tener en cuenta que el BPTT puede enfrentar problemas de desvanecimiento o explosión de gradientes, especialmente en secuencias largas. Para abordar estos problemas, como ya se ha mencionado previamente, se propone el uso de arquitecturas más complejas, como las LSTM y las GRU.

Es relevante destacar que el BPTT puede volverse numéricamente inestable cuando las funciones de activación son no derivables o tienen una derivada computacionalmente compleja. Por esta razón, como se verá en la sección 3.2.4, es importante que estas funciones de activación sean fácilmente derivables.

Además, el BPTT puede ser computacionalmente costoso debido a la gran cantidad de parámetros que suelen tener las redes neuronales. Por esta razón, se han desarrollado diferentes optimizadores que ayudan al algoritmo de retropropagación a través del tiempo a converger hacia ese mínimo de manera más eficiente. En este trabajo, nos centraremos específicamente en el optimizador conocido como Descenso de Gradiente Estocástico (SGD).

3.2.3. Descenso del gradiente estocástico (SGD)

El optimizador SGD pertenece a la familia de optimizadores que utilizan la técnica del gradiente descendente. Es un algoritmo iterativo que busca el mínimo actualizando los parámetros de la función en la dirección opuesta al gradiente. Esta técnica solo requiere el cálculo de las derivadas de primer orden de la función con respecto a los parámetros. Sin embargo, en redes neuronales con muchos parámetros, el cálculo puede ser computacionalmente costoso. Por lo tanto, en la práctica, se divide el conjunto de datos en lotes más pequeños para procesarlos por separado. Aquí es donde entra en juego el concepto de función estocástica, que se compone de una suma de subfunciones evaluadas en diferentes lotes formados a partir del conjunto de datos.

Para comprender cómo funciona este proceso, es importante entender tres términos clave:

- **Época:** Una época o *epoch* es el número de veces que el algoritmo procesa todo el conjunto de datos o el conjunto de entrenamiento completo si se ha dividido en lotes.
- **Lote:** Un lote o *batch* es cada una de las submuestras en las que se divide el conjunto de datos o entrenamiento. El tamaño del lote determina cuántos elementos del conjunto de datos se procesan en cada paso de aprendizaje del algoritmo.
- **Iteración:** Una iteración es cada vez que el algoritmo procesa un lote en cada época.

Si el conjunto de entrenamiento se divide en varios lotes, cada época del algoritmo puede contar con varias iteraciones. En particular, para cada una de las épocas fijadas, se cumple la siguiente condición:

$$\text{Tamaño de lote} \times \text{número de iteraciones} \geq \text{tamaño del conjunto de entrenamiento.}$$

Debido a esto, solo se necesita definir un elemento adicional además del número de épocas, ya que el otro queda determinado de manera única.

El SGD es un método iterativo que actualiza los pesos θ utilizando un único elemento seleccionado aleatoriamente del conjunto de entrenamiento, siguiendo la siguiente regla:

$$\theta_{k+1} = \theta_k - \eta \nabla_\theta \mathscr{L}(\theta_k), \tag{3.38}$$

donde η es una constante positiva conocida como tasa de aprendizaje, que regula el grado de cambio en la dirección opuesta al gradiente en cada paso, y k la iteración k-ésima.

El algoritmo de descenso de gradiente estocástico estándar (SGD) presenta problemas en términos de velocidad y estabilidad de convergencia. Para solucionar estos problemas, existen varias adaptaciones. Una de ellas es la inclusión del *momentum*, que consiste en agregar una fracción del vector de actualización del paso anterior al vector de actualización actual. Este método sigue la siguiente regla:

$$\begin{aligned} v_k &= \lambda v_{k-1} + \eta \nabla_\theta \mathscr{L}(\theta_k), \\ \theta_{k+1} &= \theta_k - v_k. \end{aligned} \tag{3.39}$$

donde λ es el término del *momentum*, que generalmente tiene un valor de 0.9 y v_k es la estimación del momento de primer orden del gradiente en la iteración k.

Otra mejora del algoritmo SGD es utilizar el gradiente acelerado de Nesterov, que introduce una forma de previsión en la actualización de los pesos [21]. La idea detrás de Nesterov es calcular una aproximación de la posición futura de los parámetros antes de calcular el gradiente en esa posición aproximada.

En el método estándar de SGD, el gradiente se calcula en la posición actual de los parámetros y se utiliza para actualizar los pesos. En cambio, en Nesterov, se utiliza el término de *momentum* para mover

los parámetros a una posición aproximada de su próxima actualización. Luego, se calcula el gradiente en esa posición aproximada y se utiliza para realizar la actualización final de los pesos.

La fórmula para el gradiente acelerado de Nesterov es la siguiente:

$$\begin{aligned} v_k &= \lambda v_{k-1} + \eta \nabla_\theta \mathscr{L}(\theta_k - \lambda v_{k-1}), \\ \theta_{k+1} &= \theta_k - v_k. \end{aligned} \tag{3.40}$$

Al utilizar el gradiente en la posición aproximada, Nesterov permite que el algoritmo tenga una visión anticipada y considere la dirección del gradiente en esa posición antes de realizar la actualización de los pesos. Esto ayuda a evitar oscilaciones innecesarias y permite un movimiento más suave hacia el mínimo de la función de pérdida.

Dentro de la familia de los optimizadores que utilizan la técnica del gradiente descendente, también se incluye el descenso de gradiente en mini-lotes (*mini-batch SGD*). A diferencia del SGD, que actualiza los parámetros después de procesar cada elemento del conjunto de entrenamiento, el descenso de gradiente en mini-lotes actualiza los parámetros después de procesar un lote completo del conjunto de entrenamiento. La regla de actualización para el descenso de gradiente en mini-lotes es similar a la del SGD estándar, pero en este caso, se evalúa la función de pérdida en un lote que consta de n muestras, donde n representa el tamaño del mini-lote. Este valor es un hiperparámetro que se elige según las características del problema y las capacidades computacionales disponibles.

El uso del descenso de gradiente en mini-lotes combina eficiencia computacional, estabilidad de convergencia y capacidad de adaptación del modelo, lo que lo convierte en una opción mejor que el SGD estándar.

3.2.4. Funciones de activación en la etapa de aprendizaje

En el proceso de aprendizaje de una red neuronal, la elección de las funciones de activación en cada capa es crucial para garantizar la derivabilidad de la función de pérdida y evitar problemas en el cálculo de los gradientes. A continuación, analizaremos las ventajas y desventajas de diferentes funciones de activación:

- **Función sigmoide:** La función sigmoide es una función de activación no lineal que devuelve valores en el rango [0, 1]. Su principal ventaja es que es fácilmente diferenciable:

$$\begin{aligned} f(x) &= \frac{1}{1+e^{-x}}, \\ f'(x) &= \frac{e^{-x}}{(1+e^{-x})^2} = f(x)(1-f(x)). \end{aligned} \tag{3.41}$$

  Sin embargo, la función sigmoide puede presentar el problema del gradiente desvaneciente, debido a que la derivada se vuelve casi nula para entradas muy grandes o muy negativas. Esto puede dificultar la convergencia y el aprendizaje de la red.

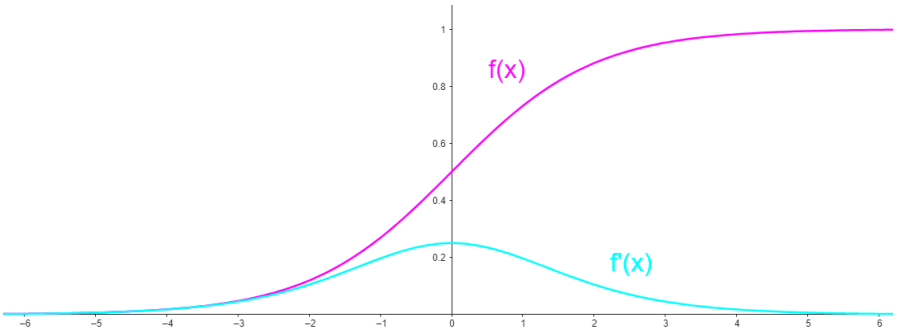

Figura 3.9: Función sigmoide y su derivada.

- **Función de tangente hiperbólica:** Como alternativa a la función sigmoide, se puede utilizar la función tangente hiperbólica (3.7), que tiene la ventaja de ser una función centrada en cero y cuya derivada también depende de la función:

$$f'(x) = \frac{(e^x + e^{-x})^2 - (e^x - e^{-x})^2}{(e^x + e^{-x})^2} = 1 - f^2(x). \tag{3.42}$$

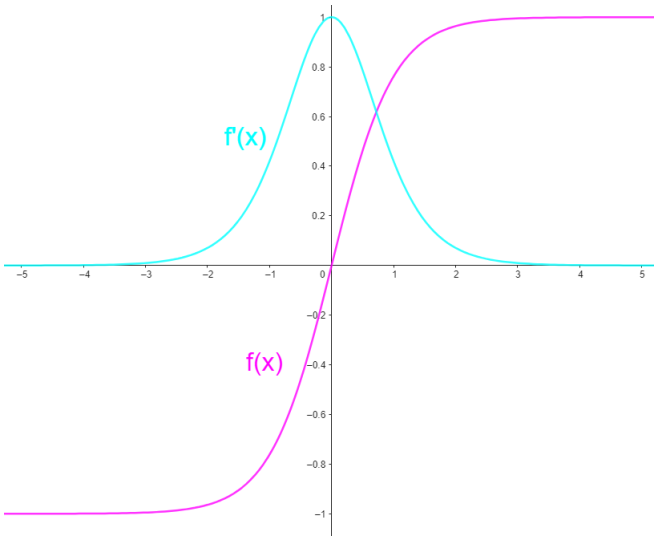

Figura 3.10: Función tangente hiperbólica y su derivada.

Al igual que la función sigmoide, la función tangente hiperbólica puede sufrir el problema del gradiente desvaneciente debido a que los valores del gradiente se aproximan a cero en ciertas regiones.

- **Función ReLU:** Para abordar el problema del gradiente desvaneciente, se introdujo la función ReLU (*Rectified Linear Unit*), que se define como (3.5).

  Aunque la función ReLU no es diferenciable en $x = 0$, se puede definir su derivada como:

$$f'(x) = \begin{cases} 1 & si \quad x > 0. \\ \\ 0 & si \quad x \leq 0. \end{cases} \tag{3.43}$$

La función ReLU evita el problema del gradiente desvaneciente para valores positivos, pero puede

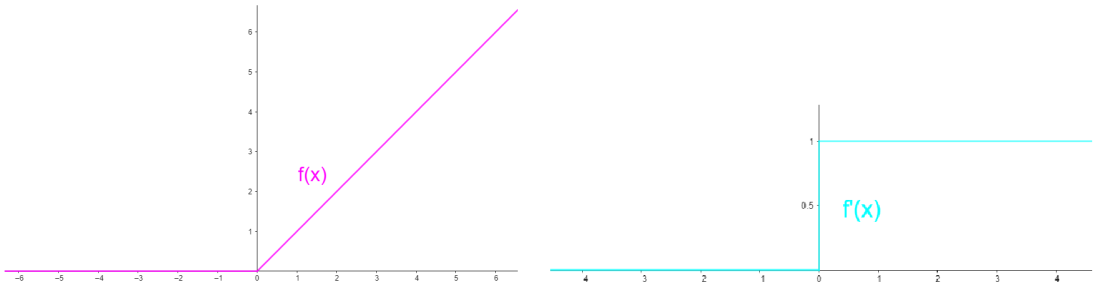

Figura 3.11: Función ReLU (izquierda) y su derivada (derecha).

causar el fenómeno de "neuronas muertas" cuando los valores de entrada son negativos, lo que significa que la neurona no se activa y no contribuye al aprendizaje. A pesar de este problema, la función ReLU es ampliamente utilizada en las capas ocultas de las redes debido a su simplicidad y eficiencia computacional.

- **Función ELU:** La función ELU se utiliza como alternativa a la función ReLU para solucionar el problema de las "neuronas muertas". Se define como (3.6) y su derivada se calcula como:

$$f'(x) = \begin{cases} 1 & si \quad x > 0. \\ \\ \alpha e^x & si \quad x \leq 0. \end{cases} \tag{3.44}$$

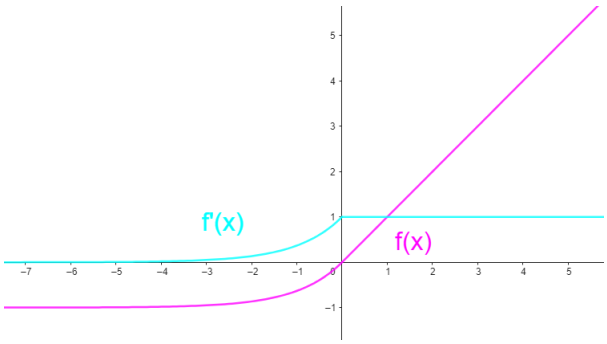

Figura 3.12: Función ELU y su derivada para $\alpha = 1$.

La función ELU evita las "neuronas muertas" para valores negativos, lo que ayuda a mantener el flujo de información en la red neuronal. Sin embargo, tiene la limitación de no estar centrada en cero y puede producir valores muy altos para entradas grandes, lo que puede llevar a una explosión del modelo.

- **Función *softmax*:** La función *softmax* se utiliza en las capas de salida de las redes neuronales para obtener una distribución de probabilidad sobre un vector $\mathbf{x} \in \mathbb{R}^n$. Se define como:

$$f(\mathbf{x}) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}, \tag{3.45}$$

lo que significa que devuelve un valor entre 0 y 1. La función *softmax* se utiliza principalmente en problemas de clasificación multiclase, ya que proporciona las probabilidades de pertenencia a cada clase del modelo. A diferencia de otras funciones de activación, la función *softmax* no suele presentar problemas en el proceso de aprendizaje de la red.

Las funciones de activación desempeñan un papel crucial en el éxito de los modelos de aprendizaje profundo al introducir no linealidad en las redes neuronales. Sin embargo, a medida que las redes se vuelven más profundas y complejas, pueden surgir desafíos relacionados con la estabilidad y variabilidad de las distribuciones de activaciones en cada capa. Por ello, surge la *batch normalization* como solución clave para enfrentarse a estos problemas.

3.2.5. *Batch normalization*

La *batch normalization* es una técnica utilizada para mejorar el entrenamiento de una red neuronal y acelerar su convergencia. Consiste en normalizar cada lote de datos de entrenamiento al ajustar la media y la varianza de las activaciones. El objetivo principal de la normalización por lotes es evitar que las activaciones se encuentren en rangos muy diferentes entre sí.

Al normalizar los datos, se logra que las distancias entre ellos estén en un rango más compacto, generalmente entre 0 y 1. Esto beneficia al modelo, ya que los datos están más cerca unos de otros, lo que facilita la creación de una línea de decisión más clara. La normalización por lotes también ayuda a mitigar el impacto de la inicialización de los pesos y mejora la estabilidad del proceso de entrenamiento.

Al utilizar la *batch normalization*, las redes neuronales tienden a aprender más rápidamente y se vuelven más robustas frente a cambios en los datos de entrada. Además, la normalización por lotes puede actuar como una regularización al introducir cierta aleatoriedad en las activaciones de cada lote durante el entrenamiento.

Es importante ajustar adecuadamente los hiperparámetros de la normalización por lotes, como el factor de normalización, para obtener los mejores resultados en el problema específico que se esté abordando.

Sin embargo, incluso cuando se utilizan las mejores técnicas de modelado, existe un desafío común que puede comprometer la precisión y la capacidad de generalización del modelo, el sobreajuste u *overfitting*. A continuación, exploraremos en detalle el concepto de sobreajuste, sus causas y las estrategias para mitigarlo.

3.2.6. Sobreajuste

El sobreajuste se produce cuando una red neuronal aprende en exceso los detalles y peculiaridades de su conjunto de datos de entrenamiento, lo que lleva a una falta de generalización para otros datos futuros. Esto significa que el modelo se ajusta demasiado bien a los datos específicos del conjunto de entrenamiento y pierde la capacidad de aplicarse de manera efectiva a datos nuevos. Para abordar el sobreajuste, se suelen aplicar técnicas de regularización, como el *dropout*.

La técnica de *dropout* se utiliza para prevenir el sobreajuste y mejorar la capacidad de generalización en redes neuronales. Consiste en desactivar aleatoriamente un número de neuronas de la red durante cada iteración del entrenamiento.

El objetivo principal del *dropout* es evitar que las neuronas dependan en exceso de otras neuronas específicas. Al desactivar un conjunto aleatorio de neuronas en cada iteración, se introduce una especie de ruido en el proceso de entrenamiento, lo que obliga a las neuronas restantes a aprender características más robustas e independientes. Esto evita la memorización de patrones específicos y promueve la detección de patrones más amplios y relevantes.

La capa de *dropout* se coloca típicamente después de las capas de activación. La tasa de *dropout* (*dropout rate*) determina la fracción de neuronas que se desactivan en cada iteración. Es importante ajustar

adecuadamente la tasa de *dropout*, ya que un valor muy alto puede llevar a una pérdida de información importante, mientras que un valor muy bajo puede no tener un efecto significativo en la prevención del sobreajuste.

Además de aplicar esta técnica para evitar el sobreajuste, es recomendable implementar un *checkpointer* para guardar el modelo en el punto donde mejor se desempeña en el conjunto de validación durante el proceso de entrenamiento. Este enfoque asegura que si el modelo comienza a sobreajustarse a los datos de entrenamiento y su rendimiento en el conjunto de validación empeora, podamos revertir al último punto de control almacenado, evitando entrenar en una dirección perjudicial. La incorporación de esta técnica nos brinda la ventaja de asegurar que el modelo final seleccionado no esté sobreajustado y pueda alcanzar el mejor rendimiento posible en datos no vistos.

3.3. Validación de la red

Una vez revisados los componentes de una BCRNN y su proceso de aprendizaje, es necesario definir una métrica que permita evaluar de manera rigurosa el rendimiento del modelo en nuevos datos. Para esta tarea, se utilizará el conjunto de validación.

En general, la mayoría de los criterios de validación se centran en medir la precisión del modelo. Sin embargo, dependiendo del proyecto en el que estemos trabajando, también puede ser importante enfocarse en una ejecución rápida del modelo o en modelos que utilicen menos memoria, incluso si la precisión disminuye ligeramente.

En nuestro caso particular, nos interesa evaluar la precisión del modelo en la transcripción de audio a texto. Para esta tarea, utilizaremos dos métricas comunes: *Word Error Rate* (WER) y *Character Error Rate* (CER).

3.3.1. *Word Error Rate* (WER)

La métrica WER es ampliamente utilizada en la evaluación de los sistemas de ASR [4, 20]. Esta métrica evalúa la precisión del modelo a nivel de palabras y se basa en el cálculo del número mínimo de inserciones, eliminaciones y sustituciones necesarias para transformar una frase hipotética, que representa la frase predicha, en una frase de referencia, que representa el etiquetado verdadero. El cálculo del WER deriva de la distancia de Levenshtein, que mide la distancia entre dos secuencias [29].

La fórmula para el cálculo del WER es la siguiente:

$$WER = \frac{S+D+I}{N} \cdot 100 = \frac{S+D+I}{S+D+C} \cdot 100. \tag{3.46}$$

Donde

- S es el número de sustituciones necesarias para transformar la frase hipotética en la frase de referencia.
- D es el número de eliminaciones necesarias para transformar la frase hipotética en la frase de referencia.
- I es el número de inserciones necesarias para transformar la frase hipotética en la frase de referencia.
- C es el número de palabras correctas en la frase hipotética.
- N es el número máximo entre el total de palabras de la frase de referencia y el total de palabras en la frase predicha.

El WER se expresa como un porcentaje y proporciona una medida de lo bien que el modelo puede transcribir correctamente una frase en comparación con la frase de referencia.

Además, podemos calcular la tasa de acierto (*Word Accuracy*, WAcc) como:

$$WAcc = 1 - WER = \frac{C - I}{N} \cdot 100. \tag{3.47}$$

Es importante destacar que esta métrica no tiene en cuenta el efecto que los diferentes tipos de errores pueden tener sobre la probabilidad de obtener un resultado exitoso. Por ejemplo, se puede considerar que un error de sustitución es más grave que un error de inserción, siendo posible modificar el WER para asignar mayor o menor valor a determinados tipos de errores.

3.3.2. *Character Error Rate* (CER)

El CER es otra métrica fundamental para evaluar el rendimiento de los sistemas de ASR [22, 6]. A diferencia del WER, el CER se enfoca en medir la precisión a nivel de caracteres. La fórmula para calcular el CER es similar a la del WER, pero en lugar de considerar palabras, se comparan los caracteres individuales en la transcripción del modelo con los caracteres de referencia.

$$CER = \frac{S + D + I}{N} \cdot 100. \tag{3.48}$$

Donde:

- S es el número de caracteres sustituidos o cambiados incorrectamente en la frase hipotética en comparación con los caracteres de referencia.
- D es el número de caracteres omitidos o no transcritos en la frase hipotética.
- I es el número de caracteres insertados incorrectamente en la frase hipotética.
- N es el número máximo entre el total de caracteres de la frase de referencia y el total de caracteres en la frase predicha.

Al igual que el WER, el CER se expresa como un porcentaje y proporciona una medida de lo bien que el modelo puede transcribir correctamente los caracteres en comparación con el texto de referencia. También permite calcular la tasa de acierto (*Character Accuracy*, CAcc) como:

$$CAcc = 1 - CER. \tag{3.49}$$

Tanto el WER como el CER proporcionan una evaluación cuantitativa del rendimiento del modelo en términos de precisión de transcripción. Es importante tener en cuenta que, en el resto del trabajo, cuando nos refiramos a WER o CER, estaremos hablando del valor promedio de todas las muestras del conjunto. Destacar que un menor porcentaje de WER y CER corresponde a un mejor rendimiento del modelo desarrollado.

Estas métricas son esenciales para identificar áreas de mejora en el sistema y lograr una mayor precisión en la transcripción. Dependiendo del contexto del proyecto, se puede dar prioridad a una u otra métrica, o incluso combinarlas, para obtener una evaluación más completa del modelo.

Capítulo 4

Creación y evaluación del proyecto

En este capítulo, se presenta el estudio y los resultados del proyecto, que se enfoca en la implementación de técnicas de Inteligencia Artificial para la conversión de audio a texto. La solución propuesta se basa en la aplicación de los conceptos teóricos sobre redes neuronales explicados previamente en la memoria.

Inicialmente, el objetivo consistía en desarrollar un programa para la gestión de incidencias que permitiera identificar el tipo de incidencia presentada mediante la transcripción de audio a texto. Sin embargo, durante la investigación, surgieron diversas complicaciones que se han intentado abordar y resolver.

La dificultad principal encontrada fue el entrenamiento del modelo, que demandaba un conjunto de datos considerablemente grande y específicamente creado para la identificación de incidencias. La elaboración de dicho conjunto de datos implicaba una inversión de tiempo significativa, lo que llevó a descartar este enfoque inicial. En su lugar, se adoptó una nueva estrategia que implicaba aplicar el programa en un conjunto de datos de menor tamaño. Esta alternativa se eligió con el objetivo de evaluar la viabilidad de desarrollar un sistema de conversión de audio a texto aplicable a la gestión de incidencias.

El nuevo enfoque se centró en lograr la transcripción automática de audios, lo que requirió la creación de un conjunto de datos de entrenamiento adecuado. A continuación, describimos en detalle la construcción de este conjunto de datos.

4.1. Elaboración del conjunto de datos

La creación de un conjunto de datos de calidad es esencial en este trabajo, dado que la efectividad de los algoritmos de *deep learning* depende en gran medida de la calidad y cantidad de los datos utilizados. Es importante destacar que la elaboración de este conjunto de datos está limitada por la necesidad de rapidez en su realización.

En un principio, se consideró generar manualmente cada muestra del conjunto de datos, grabando personalmente los audios. Sin embargo, este enfoque era demasiado lento y laborioso, por lo que se optó por una estrategia automatizada. Esta estrategia consistió en generar automáticamente el conjunto de datos utilizando el contenido de libros sin derechos de autor. Se diseñaron scripts que, dado un libro, lo segmentaban en frases y luego un programa automático leía cada frase y la almacenaba en formato de audio. Esta segmentación se realizaba avanzando a la siguiente frase cada vez que se encontraba un punto, un punto y coma, dos puntos, o una exclamación o interrogación de cierre en el texto original.

El proceso de generación del conjunto de datos se realizó de la siguiente manera:

- Se seleccionó un libro sin derechos de autor, en este caso, "El Quijote", y se dividió en frases para su posterior procesamiento.
- Cada una de estas frases se limpió, eliminando todos los signos que la red no es capaz de reconocer. Dado que la red no es capaz de identificar letras mayúsculas, todas las frases se convirtieron a minúsculas.
- Se utilizó un programa de lectura automática llamado gTTS (*Google Text-to-Speech*) [10], que es una biblioteca de *Python* y una herramienta CLI (*Command-Line Interface*) que interactúa con la API de texto a voz de *Google Transalte*. Esta biblioteca generó archivos de audio en formato *MP3* a partir de las frases procesadas utilizando la voz de *Google Transalte*.
- La salida del programa se redirigió a un archivo para guardar los audios generados en formato *MP3*.
- Los archivos de audio en formato *MP3* se convirtieron al formato *WAV*, debido a que *Python* trabaja mejor con este tipo de formato de audio.
- Finalmente, se normalizó la frecuencia de muestreo de los archivos de audio a 16.000 Hz para su procesamiento en *Python*

Utilizando esta estrategia, se logró generar un conjunto de datos con 14.554 muestras. Este conjunto de muestras se dividió en dos subconjuntos: el conjunto de entrenamiento, que representa el 90 % de las muestras, y el conjunto de test o validación, que abarca el 10 % restante.

Con el propósito de representar estas muestras, se generó un archivo JSON para cada subconjunto, el cual almacena información esencial sobre cada muestra. Cada entrada en el archivo JSON incluye la duración del audio, su transcripción textual y la ruta de almacenamiento en el ordenador. Recordar que cada muestra se introduce en el modelo como un par de datos, compuesto por una secuencia de características extraídas del audio y su correspondiente transcripción textual. Por lo tanto, este archivo JSON juega un papel fundamental al cargar las muestras en el entorno de ejecución de *Python*.

Para extraer las características de los audios, se han utilizado ventanas de Hann con un tamaño de 20 milisegundos y un desplazamiento entre ventanas sucesivas de la mitad del tamaño de la ventana para lograr un solapamiento parcial.

Además, decidimos entrenar los modelos con espectrogramas y MFFC. En caso de utilizar espectrogramas, la longitud del vector de entrada es 161, representando la cantidad de puntos en la escala de frecuencia que estarán presentes después de aplicar la transformación de dominio de frecuencia. Por otro lado, si se opta por utilizar los coeficientes MFCC, la longitud del vector es 13, como es común en el procesamiento de señales de audio [19].

Las trascripciones textuales de los audios se convierten a secuencias numéricas a través de un proceso de etiquetado. Para llevar a cabo este etiquetado se estableció el conjunto L de clases o etiquetas de la red. Este conjunto está compuesto por cada una de las letras del alfabeto castellano, junto con el espacio en blanco, denotado por ε, y las vocales con tilde, lo que resulta en un total de 33 etiquetas. Para etiquetar cada transcripción textual, se asigna a cada etiqueta del conjunto L un número, y la transcripción textual de cada frase se codifica mediante una secuencia numérica, reflejando los números de las etiquetas del conjunto L presentes en la frase. A continuación, se presenta un ejemplo ilustrativo para entender mejor este proceso.

Ejemplo 5 (Etiquetado)**.** Suponemos que tenemos el conjunto de etiquetas $L = \{\varepsilon, a, d, h, l, m, n, o, u\}$ y la asignación:

$$\varepsilon = 0\ , a = 1\ ,\ d = 2\ , h = 3\ , l = 4\ , m = 5\ , n = 6\ , o = 7\ , u = 8.$$

Entonces la frase="hola mundo" se codifica en secuencia numérica como (3741058627). □

Para revertir este proceso y convertir una secuencia de números generada por la red en texto, se realiza el proceso inverso. Cada número en la secuencia se corresponde con su etiqueta correspondiente en el conjunto L, permitiendo así obtener la transcripción textual correspondiente.

Una vez explicado detalladamente cómo se creó el conjunto de muestras utilizado para el entrenamiento de la red y cómo se llevó a cabo su etiquetado, procedemos a describir los modelos de redes utilizados en el proceso de transcripción de audio a texto.

4.2. Creación del modelo

Como mencionamos previamente, el conjunto de entrenamiento inicial consta de 13.099 muestras, y el conjunto de validación de 1.455 muestras. Sin embargo, con el objetivo de mejorar la calidad de los datos y evitar posibles errores durante el entrenamiento de la red, se decidió eliminar aquellas muestras con una duración de audio superior a 30 segundos. Tras este filtrado, el conjunto de entrenamiento quedó con 12.717 muestras, y el conjunto de validación con 1.430 muestras.

Para abordar el problema de transcripción de audio a texto, se seleccionó una red BCRNN debido a sus diversos beneficios, como se explicó en el capítulo anterior. La estructura de la red utilizada en este trabajo se muestra en la Figura 3.1 y consta de los siguientes bloques:

1. **Bloque de convolución**: Esta capa convolucional se encarga de extraer características relevantes de los datos de entrada. Después de la capa convolucional, se aplicó una capa *batch normalization* para mejorar la estabilidad y acelerar el entrenamiento de la red.

2. **Bloques recurrentes bidireccionales**: La red BCRNN utiliza tres bloques recurrentes bidireccionales. Cada bloque incluye una capa recurrente bidireccional que utiliza neuronas GRU. Recordar que la elección de usar neuronas GRU se fundamenta en su menor coste computacional en comparación con las LSTM, sin que esto afecte significativamente a la eficiencia del modelo. Después de cada capa recurrente bidireccional, se aplicó una capa *dropout* para prevenir el sobreajuste y una capa *batch normalization* para la normalización de los datos.

3. **Capa de salida**: Después de los bloques recurrentes bidireccionales, se agregó una capa de salida. Se utilizó una capa de tiempo distribuido (*time distributed*) seguida de una capa densa con activación *softmax*, definida en (3.45). La capa de tiempo distribuido permite aplicar una capa densa a cada paso de tiempo de la secuencia generada por los bloques recurrentes. La activación *softmax* se utiliza para producir las salidas de clasificación multiclase, asignando una probabilidad a cada clase del conjunto L.

En resumen, la estructura de la red BCRNN implementada consta de una capa convolucional para la extracción de características, tres bloques recurrentes bidireccionales para aprender dependencias a largo plazo en los datos secuenciales, y una capa de salida para producir las salidas de clasificación. En cada una de estas capas, se aplicó una semilla para inicializar los pesos aleatoriamente a partir de una distribución normal, garantizando la reproducibilidad en la inicialización de los pesos.

Dado que el entrenamiento de modelos es un proceso iterativo cuyo objetivo principal es encontrar los hiperparámetros óptimos con el fin de minimizar el error, hemos generado una serie de modelos variando los parámetros utilizados en la red BCRNN. A continuación, detallamos los modelos creados y sus características específicas.

4.2.1. Primer modelo

En el primer modelo, se utilizó una capa convolucional con 5 filtros de tamaño 3, un *stride* de 2 y la función de activación ReLU. Además, se aplicó *padding* para mantener el tamaño de salida de la capa convolucional. Las capas recurrentes bidireccionales constaron de 15 neuronas GRU con función de activación ReLu.

Al compilar el modelo, se utilizó la función de pérdida CTC y el optimizador SGD con una tasa de aprendizaje $\eta = 0{,}005$, *momentum* $\lambda = 0{,}9$ y gradiente acelerado Nesterov.

En cuanto al tipo de características utilizadas como datos de entrada a la red, se entrenaron dos versiones del modelo, una utilizando MFCC y otra utilizando espectrogramas.

Modelo 1 entrenado con MFCC

Esta versión del modelo fue entrenada durante 20 épocas, utilizando lotes de 20 muestras, resultando en 635 iteraciones por época. El tiempo de entrenamiento por cada época fue aproximadamente 25 minutos, y el número total de parámetros fue 11.804.

La Figura 4.1 muestra la curva de aprendizaje del primer modelo entrenado con MFCC. Esta gráfica ilustra la evolución de la pérdida de entrenamiento y validación a lo largo de las épocas, permitiendo visualizar cómo el modelo aprende a medida que se expone a más datos y brindando información sobre su comportamiento tanto en los datos de entrenamiento como en los datos no vistos del conjunto de validación.

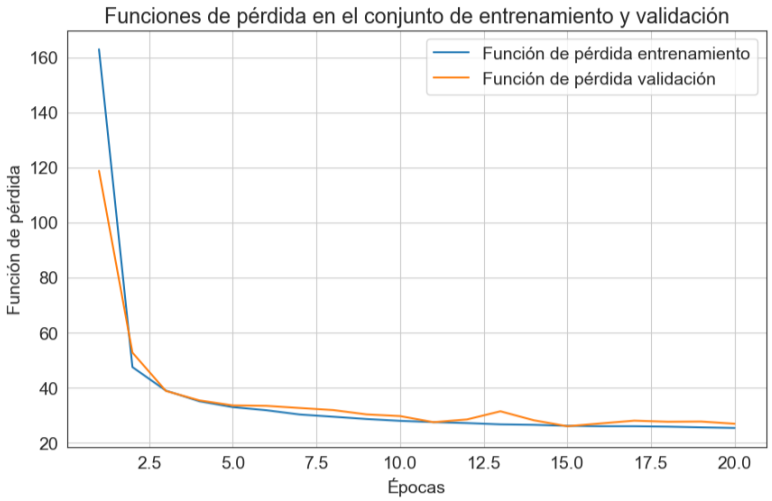

Figura 4.1: Curva de aprendizaje con espectrograma del primer modelo con MFCC.

Durante el entrenamiento del modelo, se observa una disminución de la pérdida tanto en los datos de entrenamiento como en los de validación a media que avanzan las épocas. Esto sugiere una tendencia general positiva en el rendimiento del modelo. En otras palabras, el modelo ha aprendido bien los patrones de entrenamiento y generaliza adecuadamente a datos no vistos.

Notar, que a partir de la época 11, la pérdida en el conjunto de entrenamiento continúa disminuyendo. En contraste, la pérdida en el conjunto de validación empieza a aumentar o mantenerse relativamente constante. Este comportamiento sugiere la posibilidad de que el modelo esté comenzando a sobreajustarse. Gracias a la implementación del *checkpointer*, podemos asegurar que el modelo final guardado, el

cual se obtiene en la época 15, no presenta sobreajuste y logra alcanzar el mejor rendimiento posible en datos no observados.

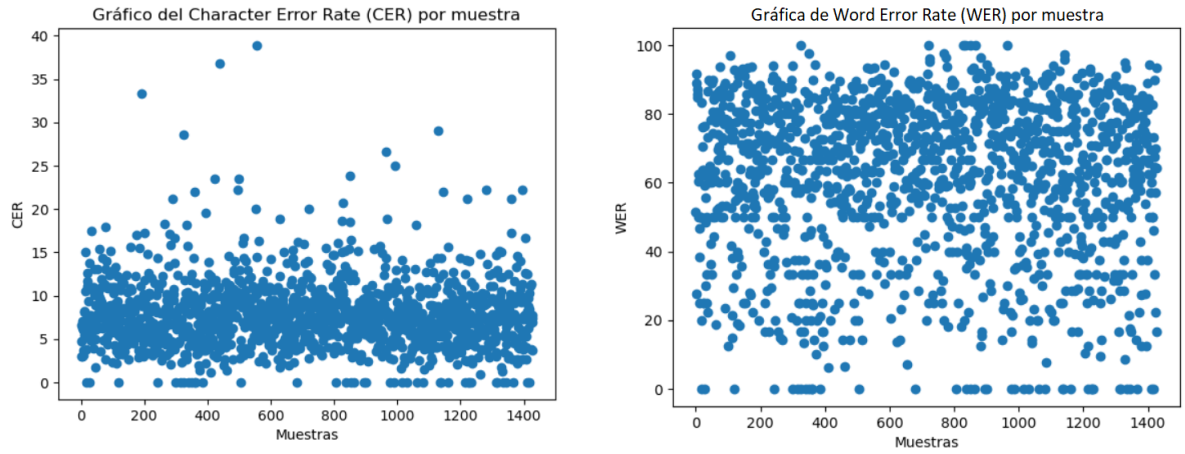

Figura 4.2: Gráfica del CER (izquierda) y del WER (derecha) en el conjunto de validación.

En el conjunto de validación, este modelo logró un CER de 7,79 % y un WER de 61,44 %, lo que se traduce en una precisión de 92,21 % en caracteres y del 38,56 % en palabras. Al observar la Figura 4.2, se puede apreciar que el CER tiende a concentrarse en un rango donde se encuentran la mayoría de las muestras, mientras que el WER presenta un comportamiento más aleatorio en las muestras. Esto se debe a que el WER es una métrica más estricta que penaliza los errores que involucran palabras completas en lugar de caracteres individuales. Veamos un ejemplo de esto.

Ejemplo 6 (Cálculo del CER y el WER)**.** Para la frase de referencia 'yo muero deseando', el modelo genera la frase 'yon muero de seando'. Para esta muestra, se obtiene un CER del 10,53 % y un WER del 50 %. Como se puede apreciar, la frase predicha no comete grandes errores, sin embargo, contiene dos palabras incorrectas, lo que resulta en un WER elevado. □

Este ejemplo ilustra cómo el WER penaliza más severamente los errores en comparación con el CER. Esta consideración, junto con un enfoque más marcado en sancionar los caracteres incorrectos en lugar de las palabras erróneas, justifica la decisión de utilizar exclusivamente el CER para evaluaciones futuras. Este mayor énfasis proviene de nuestro interés en lograr que las palabras no necesariamente se traduzcan de manera perfecta, sino que su traducción sea lo suficientemente precisa como para ser identificable.

En la gráfica del CER de la Figura 4.2 (izquierda), se observa que algunas muestras presentan un CER elevado. Al analizar el valor del CER en función de la longitud de las muestras, se observó que las muestras con un CER elevado tienden a ser aquellas con frases de longitud reducida, como se muestra en la Figura 4.3.

Este comportamiento se debe a la penalización que impone el CER a las frases cortas, ya que el cálculo se basa en la proporción de errores respecto al número total de caracteres en la frase. Para abordar esta cuestión y obtener una evaluación más justa y equilibrada del rendimiento del modelo en distintos tipos de frases, se optó por modificar esta métrica. Se implementaron dos adaptaciones para lograr una métrica más precisa en la evaluación del modelo:

1. **Ponderación de frases cortas**. Se introdujo una ponderación menor para las muestras con frases cortas (menos de 50 caracteres), mientras que se mantuvo el cálculo original para el resto de muestras. Denotamos a esta adaptación como CER_m y la calculamos como:

$$CER_m = \frac{\alpha CER_C + CER_L}{2}, \tag{4.1}$$

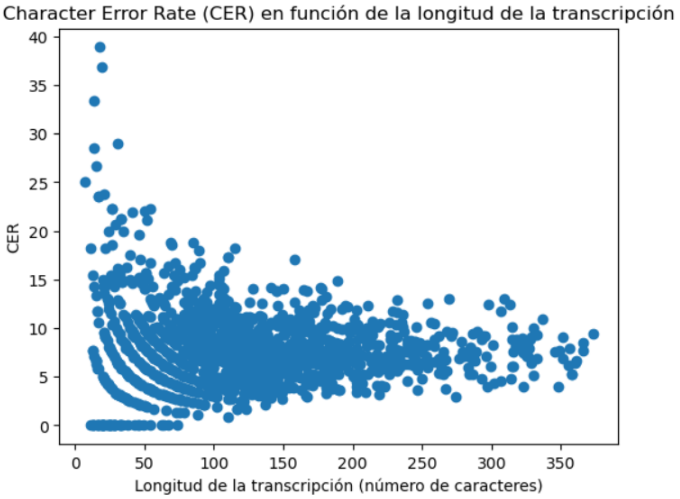

Figura 4.3: Gráfica del CER en función de la longitud de la muestra.

donde CER_C es el CER medio de las muestras con frases cortas, CER_L es el CER medio de las muestras con frases largas, y α es un parámetro que determina la importancia relativa que se otorga al error cometido en muestras con frases cortas.

2. **Ponderación por longiud de frase**. Dividimos el conjunto de muestras en 8 intervalos según la longitud en caracteres de las frases, $I_i = [0+50i, 50+50i)$, para $i = 0, ..., 6$ y $I_7 = [350, \infty)$. Luego, aplicamos una ponderación creciente a medida que aumenta el número de caracteres en las frases. Denominamos a esta adaptación como CER_M y la calculamos de la siguiente manera:

$$CER_M = \frac{\sum_{i=0}^{7} 0{,}125 \cdot (i+1) \cdot CER_i}{8}, \tag{4.2}$$

donde CER_i es el CER medio del grupo de muestras con longitud de frases en el intervalo I_i.

Ambas adaptaciones permiten una evaluación más justa y equitativa del rendimiento del modelo en diferentes frases, independiente de la longitud de estas.

En esta versión del modelo, con una ponderación $\alpha = 0{,}23$ para las frases cortas se obtuvo un CER_m de 4,80 % y un CER_M de 3,78 %. Estos valores proporcionan una evaluación más precisa y equilibrada del rendimiento del modelo, teniendo en cuenta la longitud variable de las frases en el conjunto de datos.

Modelo 1 entrenado con espectrogramas

Esta versión del modelo también fue entrenada durante 20 épocas, utilizando lotes de 20 muestras. El tiempo requerido para completar cada época fue aproximadamente 20 minutos y el número total de parámetros fue 14.024. Es importante tener en cuenta que la duración de entrenamiento no solo depende de los parámetros del modelo, sino también del rendimiento del ordenador en ese momento. En este caso, aunque esta versión del modelo tiene un mayor número de parámetros, el tiempo de entrenamiento resultó ser menor en comparación con la versión anterior.

La Figura 4.4 muestra la curva de aprendizaje del primer modelo con espectrogramas. Al igual que en la versión anterior, se puede observar una disminución gradual tanto en la pérdida de entrenamiento como en la de validación a lo largo de las épocas. Sin embargo, en esta versión, se identifican señales de sobreajuste, evidenciadas por la creciente diferencia entre la pérdida de entrenamiento y la pérdida de validación en las últimas épocas.

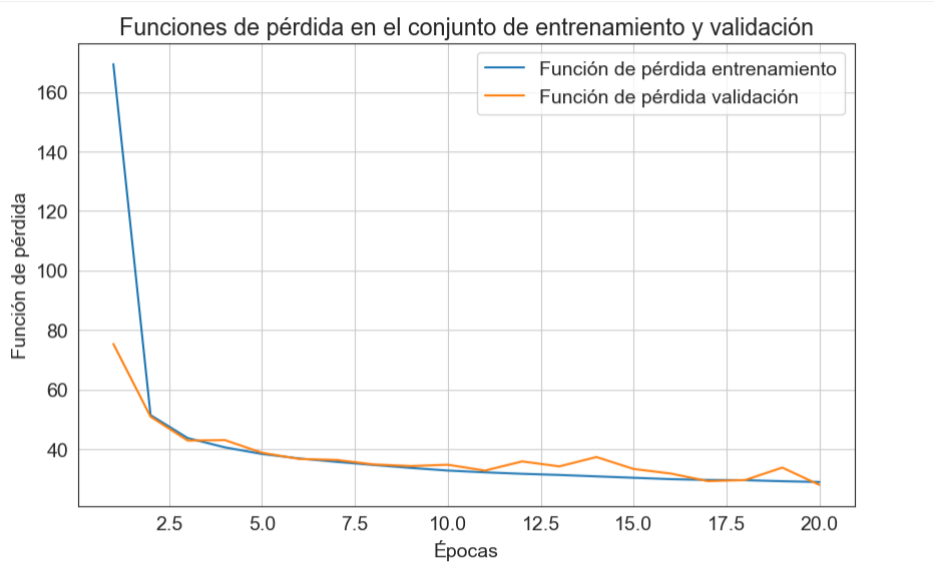

Figura 4.4: Curva de aprendizaje con espectrograma del primer modelo con MFCC.

Comparando con la versión anterior, esta muestra un comportamiento menos deseable en términos de convergencia y sobreajuste. Comienza con pérdidas más elevadas y exhibe una tendencia menos constante en la disminución de la pérdida de validación. Estos indicadores sugieren que la versión anterior podría ser más adecuada para el entrenamiento en este contexto particular.

En cuanto al rendimiento en el conjunto de validación, esta versión del modelo logró una precisión del 91,77 % en términos de caracteres, lo que es menor que la precisión obtenida en la versión anterior. Si consideramos las métricas adaptadas, se obtiene un CER_m de 5,08 % con $\alpha = 0{,}23$ y un CER_M de 3,96 %.

En resumen, podemos concluir que la versión que utiliza características MFCC como datos de entrada ha demostrado un mejor desempeño en comparación con la versión basada en espectrogramas. Con el propósito de simplificar la evaluación del rendimiento y evitar la necesidad de métricas adaptadas, se diseñó un segundo modelo que restringe la duración máxima de audio permitida. El principal objetivo de esta variante es obtener un conjunto de muestras con longitudes similares para facilitar la comparación y el análisis.

4.2.2. Segundo modelo

Para este segundo modelo, se tomó la decisión de eliminar las muestras con una duración de audio superior a 10 segundos, resultando en un conjunto de entrenamiento de 6.836 muestras y un conjunto de validación de 797 muestras.

Los parámetros utilizados en este segundo modelo son idénticos a los del primer modelo. Se empleó una capa convolucional con 5 filtros de tamaño 3, un *stride* de 2, función de activación ReLU, y se aplicó *padding*. Las capas recurrentes bidireccionales constaron de 15 neuronas GRU con función de activación ReLU. Asimismo, para compilar el modelo, se optó por la función de pérdida CTC y el optimizador SGD con una tasa de aprendizaje $\eta = 0{,}005$, *momentum* $\lambda = 0{,}9$ y gradiente acelerado Nesterov.

También se entrenaron dos versiones de este modelo, una utilizando MFCC como datos de entrada y la otra empleando espectrogramas. Ambas versiones fueron entrenadas durante 20 épocas utilizando lotes de 20 muestras, lo que resultó en 341 iteraciones por época. El tiempo de entrenamiento utilizando MFCC fue aproximadamente 1 hora y 25 minutos, mientras que con espectrogramas fue aproximadamente 1 hora y 7 minutos.

En el caso del modelo que emplea MFCC, se logró una precisión en el conjunto de validación del 91,98 % en términos de caracteres, mientras que utilizando espectrogramas se alcanzó una precisión del 90,47 %. Es importante señalar que, al igual que en el primer modelo, el uso de MFCC como características generó una mejor precisión en comparación con el uso de espectrogramas.

Realizamos un análisis comparativo de la curva de aprendizaje para evaluar los modelos. Observando la Figura 4.5, podemos notar que el segundo modelo exhibe una pérdida de validación más baja en comparación con el primer modelo, tanto cuando se utiliza MFCC como cuando se emplean espectrogramas. Este comportamiento sugiere que el segundo modelo presenta una mayor capacidad de generalización.

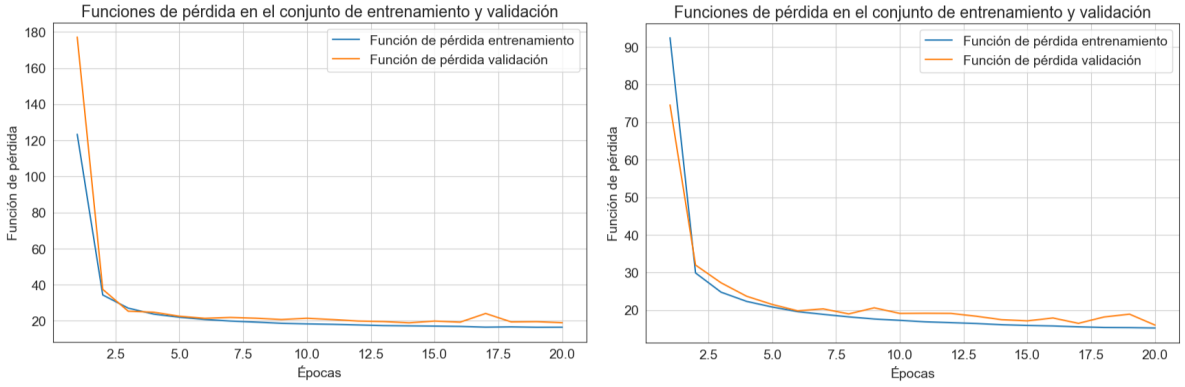

Figura 4.5: Curva de aprendizaje con MFCC (izquierda) y con espectrogramas (derecha) del segundo modelo.

En resumen, aunque el primer modelo fue entrenado con un conjunto de entrenamiento y validación más amplio, lo que podría haber contribuido a su mejor rendimiento en términos del CER, el segundo modelo muestra una mejora en su pérdida y en su rendimiento general en la tarea. Esto podría indicar una mayor capacidad de generalización y reconocimiento de caracteres en nuevos datos por parte de este segundo modelo.

Con el objetivo de seguir mejorando los resultados, se desarrolló un último modelo con un aumento en los parámetros de la red.

4.2.3. Modelo final

Después de un análisis minucioso de las limitaciones en nuestros primeros modelos, identificamos la insuficiencia de parámetros como la principal restricción. Para abordar este problema, tomamos la decisión de fortalecer la arquitectura mediante el aumento del número y tamaño de filtros en las capas convolucionales, así como incrementando el número de neuronas en las capas recurrentes.

Al trabajar con secuencias de características de audio, como MFCC o espectrogramas, es esencial reconocer la presencia de patrones temporales y frecuenciales complejos, que son fundamentales para el reconocimiento de palabras o la identificación de sonidos. Al aumentar el número de filtros en las capas convolucionales, el modelo puede detectar y aprender una variedad más amplia de características específicas en diferentes partes de la secuencia de datos.

Además, el incremento en el tamaño de los filtros en las capas convolucionales permite considerar segmentos más largos de características en los audios, lo que es especialmente beneficioso para capturar patrones que se extienden en el tiempo y que pueden abarcar múltiples ventanas temporales en el análisis de la secuencia de audio.

El aumento del número de neuronas en las capas recurrentes también es crucial al trabajar con secuencias de datos de audio. Esto permite una representación más rica y detallada de las características previamente extraídas. Dado que las características extraídas mediante MFCC o espectrogramas pueden contener información relevante sobre varios aspectos del audio, como tonalidades, ritmo y contenido semántico, contar con más neuronas permite al modelo capturar y combinar estas características de manera más efectiva. Esto, a su vez, mejora el rendimiento en tareas de clasificación, reconocimiento de patrones y análisis de audio en general.

En este modelo final, se utilizó una capa convolucional con 100 filtros de tamaño 11. El resto de los parámetros de esta capa se mantuvieron iguales que en los modelos anteriores. En las capas recurrentes bidireccionales, se incrementó el número de neuronas GRU a 100.

Entrenamos el modelo utilizando muestras con una duración máxima de 10 segundos para los audios. Esta selección está respaldada por el análisis comparativo de los dos modelos previos, así como en el tiempo excesivo necesario para entrenar el modelo con las muestras de audio de hasta 30 segundos. En este último caso, se necesitaban alrededor de 8 horas por cada época de entrenamiento.

Además, decidimos utilizar MFCC como características de entrada. Dado que el entrenamiento de este modelo requería una inversión de tiempo considerable, optamos por entrenarlo exclusivamente con MFCC como entrada a la red, considerando las mejoras observadas en los modelos anteriores. No obstante, es importante mencionar que ajustar los parámetros en el modelo entrenado con espectrogramas podría mejorar su rendimiento. En resumen, nuestras decisiones se basaron en la eficiencia del entrenamiento y en los resultados obtenidos en los modelos previos, aunque reconocemos la posibilidad de explorar otras configuraciones para obtener el mejor rendimiento en este problema específico.

En términos de compilación del modelo, se mantuvo la utilización de la función de pérdida CTC y el optimizador SGD con una tasa de aprendizaje $\eta = 0{,}005$, *momentum* $\lambda = 0{,}9$ y gradiente acelerado Nesterov.

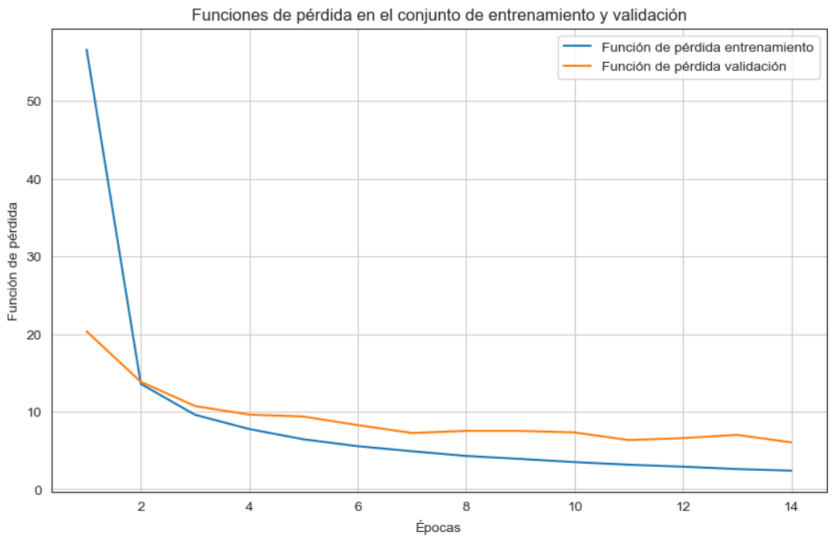

Figura 4.6: Curva de aprendizaje con espectrograma del modelo final.

El modelo fue entrenado a lo largo de 14 épocas, empleando lotes de 20 muestras. El tiempo requerido para completar cada época fue aproximadamente una hora, con un total de 505.834 parámetros.

Al analizar la curva de aprendizaje en la Figura 4.6, se puede observar una disminución constante de la pérdida a lo largo de las épocas, lo que indica un progreso positivo y una mejora en su capacidad

para comprender y procesar los datos de entrada. En comparación con los modelos anteriores, la pérdida por época es significativamente menor, lo que sugiere una adaptación más efectiva a los parámetros y un aprendizaje más óptimo a partir de los datos.

En cuanto al rendimiento en el conjunto de validación, este modelo logró una precisión del 97,19 % en términos de caracteres, lo que se traduce en un CER mucho más bajo en comparación con los modelos previos. Al observar la Figura 4.7, es evidente que las pocas muestras con valores de CER elevados corresponden principalmente a frases con una longitud reducida. Esto respalda la idea de que el CER penaliza de manera desproporcionada las frases con menor longitud. Este análisis también resalta la efectividad del modelo en la mayoría de las muestras, mostrando su capacidad para lidiar exitosamente con diversas longitudes de frases.

Figura 4.7: Gráfica del CER por muestra (izquierda) y en función de la longitud de la muestra (derecha).

En resumen, este modelo final ha superado significativamente las limitaciones de los modelos anteriores. La ampliación de parámetros, combinada con el uso de MFCC como características de entrada y un entrenamiento minucioso, ha permitido alcanzar un rendimiento sobresaliente en términos de precisión y CER. Este modelo posee la capacidad de manejar eficazmente secuencias de audio de diferentes longitudes y complejidades, demostrando un alto grado de generalización y comprensión de patrones.

Finalmente, desarrollamos e implementamos un programa en *Python* con la finalidad de corregir las transcripciones generadas por la red. Este corrector ortográfico funciona en dos fases. En la primera fase, utilizamos un modelo estadístico de corrección de errores basado en las probabilidades de palabras. Este proceso implica la creación de un vocabulario con todas las palabras presentes en el texto (El Quijote). Cuando el programa detecta una palabra traducida que no se encuentra en este vocabulario, la sustituye por la palabra con el menor número de ediciones y la mayor probabilidad en el texto original. En la segunda fase, intentamos resolver los errores de separación o unión incorrecta de palabras.

Para evaluar la efectividad del corrector, calculamos el CER medio eligiendo el valor mínimo entre el CER de la frase de referencia y la frase predicha original, y el CER de la frase de referencia y la frase predicha después de ser corregida por el programa.

Mediante la aplicación de este corrector, se logró una mejora en la precisión de tres décimas. En otras palabras, la precisión del modelo se ha incrementado a un 97,48 % en términos de caracteres en el conjunto de validación. Aunque esta mejora puede parecer pequeña, esta implementación demuestra cómo la corrección de las transcripciones mediante un programa específico puede tener un impacto positivo en la calidad y precisión general de los resultados obtenidos en el proceso de transcripción automática de audio a texto.

4.3. Conclusiones y trabajo futuro

Para concluir, los resultados presentados en este capítulo respaldan la viabilidad de construir un sistema de IA para la transcripción precisa de audio a texto. De hecho, el rendimiento del último modelo ajustado es bastante bueno en términos de precisión de caracteres.

Sin embargo, a pesar de haber alcanzado un buen nivel de precisión en la transcripción, existen diversas áreas de mejora y trabajo futuro que podrían abordarse para continuar mejorando el rendimiento del sistema:

- Ampliación del conjunto de datos: El rendimiento de los modelos de redes neuronales suele mejorar con conjuntos de datos más grandes y diversos. La recopilación de más datos de audio con una variedad de locutores, estilos de habla y acentos podría ayudar a mejorar la capacidad de generalización del modelo y su precisión en diferentes contextos.
- Reestructuración del conjunto de datos: La creación de un conjunto de datos con muestras de duraciones similares y que abarquen un vocabulario específico para la tarea, facilitaría una evaluación más precisa del rendimiento del modelo.
- Exploración de arquitecturas avanzadas: Se podría investigar la utilización de arquitecturas de modelos más avanzadas, como los transformadores, que han demostrado ser muy efectivas en tareas de sistemas de ASR.
- Aumento de parámetros y optimización: Probar con más combinaciones de parámetros y arquitecturas podría llevar a una mejora en el rendimiento.
- Desarrollo de correctores automáticos mejorados: La implementación de un corrector automático más avanzado podría mejorar aún más la precisión de las transcripciones generadas por el modelo, lo que tendría un impacto positivo en la calidad general de los resultados.
- Optimización del rendimiento computacional: Explorar formas de acelerar el entrenamiento y la inferencia del modelo utilizando hardware especializado como GPUs o TPUs podría reducir el tiempo de entrenamiento y permitir la experimentación más rápida con diferentes configuraciones.

En conclusión, aunque el modelo BCRNN actual ha logrado una buena precisión en la transcripción de audio a texto, hay muchas oportunidades para seguir mejorando su rendimiento y capacidad de adaptación a diferentes contextos y tipos de datos. Estos resultados respaldan la viabilidad de aplicar un sistema de red neuronal en la gestión de incidencias, aunque llevar a cabo una implementación efectiva en este ámbito requerirá un esfuerzo adicional, incluyendo la creación de un conjunto de datos adecuado y el posterior entrenamiento del modelo con dicho conjunto.

Bibliografía

[1] ABDEL-HAMID, O., MOHAMED, A.-R., JIANG, H., DENG, L., PENN, G., AND YU, D. Convolutional neural networks for speech recognition. IEEE Xplorer, 2014.

[2] ABUSHARIAH, M., AINON, R., ZAINUDDIN, R., ELSHAFEI, M., AND KHALIFA, O. Natural speaker-independent arabic speech recognition system based on hidden markov models using sphinx tools. Research Gate, 2010.

[3] ALBAQSHI, H., AND SAGHEER, A. Dysarthric speech recognition using convolutional recurrent neural networks. Research Gate, 2020.

[4] ALI, A., AND RENALS, S. Word error rate estimation for speech recognition: e-wer. Research Gate, 2018.

[5] CHEN, G. A gentle tutorial of recurrent neural network with error backpropagation. Research Gate, 2016.

[6] D'HARO, L. F., AND BANCHS, R. E. Automatic correction of asr outputs by using machine translation. Interspeech, 2016.

[7] GAIKWAD, S., BHARTI, W., AND YANNAWAR, P. A review on speech recognition technique. Research Gate, 2010.

[8] GOODFELLOW, I., BENGIO, Y., AND COURVILLE, A. *Deep Learning*. MIT Press, 2016.

[9] GRAVES, A., FERNÁNDEZ, S., GOMEZ, F., AND SCHMIDHUBER, J. Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural 'networks. Cs.Toronto, 2006.

[10] GTTS. gtts (google text-to-speech), a python library and cli tool to interface with google translate text-to-speech api. `https://pypi.org/project/gTTS/`.

[11] JAIN, A., ROUHE, A., GRÖNROOS, S.-A., AND KURIMO, M. Finnish asr with deep transformer models. Inter Speech, 2020.

[12] KANDA, N., YE, G., GAUR, Y., WANG, X., MENG, Z., AND YOSHIOKA, T. End-to-end speaker-attributed asr with transformer. Arxiv, 2021.

[13] KERAS. Keras: The python deep learning library. `http://keras.io/`.

[14] LOPEZ-RUIZ, R. *From Natural to Artificial Intelligence - Algorithms and Applications*. 2019.

[15] NOTEBOOK, J. The jupyter notebook. `http://jupyter.org/`.

[16] PALIWAL, K., LYONS, J., AND WOJCICKI, K. Preference for 20-40 ms window duration in speech analysis. IEEE Xplore, 2011.

[17] PLAZA, J., SANCHEZ-ZHUNIO, C., URIGÜEN, M., CORDERO, M., CEDILLO, P., AND ZAMBRANO-MARTINEZ, J. Speech recognition based on spanish accent acoustic model. Research Gate, 2022.

[18] PÉREZ, M., AND RUIZ, F. Procesamiento de la señal y de la imagen, 2017. Apuntes de la asinatura PSI del Departemento de Matemáticas, Facultad de Ciencias, Unizar.

[19] RAKIBUL, H., AND ZAKIR, H. How many mel-frequency cepstral coefficients to be utilized in speech recognition? a study the bengali language. IET, 2021.

[20] ROY, S. Semantic-wer: A unified metric for the evaluation of asr transcript for end usability. Research Gate, 2021.

[21] RUDER, S. An overview of gradient descent optimization algorithms. Arxiv, 2016.

[22] SARI, L., HASEGAWA-JOHNSON, M., AND YOO, C. D. Counterfactually fair automatic speech recognition. IEEE, 2021.

[23] SCHMIDT, R. Recurrent neural networks (rnns): A gentle introduction and overview. Arxiv, 2019.

[24] SHEWALKAR, A. N. Comparison of rnn, lstm and gru on speech recognition data. Semantic Schoolar, 2018.

[25] SULISTYANINGSIH, S., PUTRANTO, P., QURRACHMAN, T., DESVASARI, W., DAUD, P., WIJAYANTO, Y. N., MAHMUDIN, D., KURNIADI, D., RAHMAN, A. N., HARDIATI, S., SETIAWAN, A., DARWIS, F., AND PRISTIANTO, E. Performance comparison of blackman, bartlett, hanning, and kaiser window for radar digital signal processing. IEEE Xplore, 2019.

[26] TENSORFLOW. An end-to-end open source machine learning platform. `http://www.tensorflow.org/`.

[27] TOMBALOĞLU, B., AND ERDEM, H. Turkish speech recognition techniques and applications of recurrent units (lstm and gru). DergiPark, 2021.

[28] VINYALS, O., RAVURI, S. V., AND POVEY, D. Revisiting recurrent neural networks for robust asr. IEEE Xplorer, 2012.

[29] WIKIPEDIA. Distancia de levenshtein. Wikipedia.

[30] ZHANG, Z. Derivation of backpropagation in convolutional neural network (cnn). GitHub, 2016.

[31] ZHAO, H., YUFENG, X., HAN, J., AND ZHANG, Z. Compact convolutional recurrent neural networks via binarization for speech emotion recognition. IEEE Xplorer, 2019.