UNIVERSIDAD AUTÓNOMA DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería de Tecnologías y Servicios de Telecomunicación

TRABAJO FIN DE GRADO

Detección de Actividad de Voz basada en Redes Neuronales

Patricia Alonso de Apellániz
Tutor: Alicia Lozano Díez
Ponente: Joaquín González Rodríguez

Julio 2018

Detección de Actividad de Voz basada en Redes Neuronales

AUTOR: Patricia Alonso de Apellániz
TUTOR: Alicia Lozano Díez

AUDIAS – AUdio, Data Intelligence And Speech
Dpto. Tecnología Electrónica y de las Comunicaciones
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Julio 2018

Resumen (castellano)

La Detección de Actividad de Voz (*Voice Activity Detection*, VAD) se refiere a un modelo de métodos de procesamiento de una señal de audio que detecta si pequeños segmentos de esta contienen datos de voz o de no voz. Los sistemas VAD juegan un papel importante previo al procesamiento de la señal en cualquier sistema de reconocimiento, codificación o mejora de voz, ya que suele ser necesario y beneficioso el hecho de diferenciar las partes habladas de las no habladas.

Este Trabajo Fin de Grado tiene como objetivo principal explorar dos modelos distintos de redes neuronales para entrenar un sistema de Detección de Actividad de Voz. Se ha hecho una aproximación basada en una red neuronal profunda (*Deep Neural Network*, DNN) y en una red neuronal recurrente (*Recurrent Neural Network*, RNN), más específicamente una *Long Short-Term Memory*, LSTM. En ambos casos, los parámetros de entrada son las características acústicas MFCC (*Mel-frequency Cepstral Coefficients*).

Se ha utilizado como herramienta principal para la implementación de ambas redes la librería de Python, Keras, la cual actúa por encima de Theano o Tensorflow. Para el análisis y la representación de datos obtenidos a la salida se ha hecho uso de Matlab.

Los datos, tanto de entrenamiento como de validación y de test, utilizados para el análisis descrito se han tomado de la base de datos OpenSAT (*Open Speech Analytic Technologies*) desarrollada por el Instituto Nacional de Estándares y Tecnología (*National Institute of Standards and Technologies*, NIST). De esta base de datos se han tomado audios de vídeos de VAST (*Video Annotation for Speech Technologies*), audios de una operación de extinción de fuego de PSC (*Public Safety Communications*) y audios grabados de conversaciones telefónicas de IARPA Babel (*Intelligence Advanced Research Projects Activity*).

La evaluación de la mejora que pueda suponer un modelo de red frente al otro se ha hecho en base a las medidas de *accuracy* y la *Equal Error Rate,* EER, obtenidas en cada experimento. Estos valores permiten comparar el rendimiento de las distintas configuraciones de modelos de VAD que se han implementado.

Como conclusión resumida se ha obtenido un beneficio en el uso de la LSTM frente al uso de la DNN como se esperaba ya que la segunda es, en general, más adecuada para el modelado de señales temporales. Se ha demostrado también que en la tarea de VAD para el conjunto de datos utilizados funciona mejor un sistema LSTM en el que hay poca información de contexto en las secuencias de entrada, así como para el caso de las DNN, funciona mejor un modelo simple.

Palabras clave (castellano)

Detección de Actividad de Voz, Red Neuronal, DNN, LSTM, *Accuracy*, EER

Abstract (English)

Voice Activity Detection (VAD) refers to a model of methods to process an audio signal which detects whether short fragments contain voice data or not. VAD systems play an important role previous to signal processing in every voice recognition, codification or enhancement system, as it is usually necessary and beneficial to differentiate between spoken or not spoken parts.

This Bachelor Thesis focuses on exploring two different types of Artificial Neural Networks (ANN) to train a Voice Activity Detector. Two main systems are implemented: one using a Deep Neural Network (DNN) and another one using a Recurrent Neural Network (RNN), particularly a Long Short Term Memory Neural Network (LSTM). In both of them, Mel-Frequency Cepstral Coefficients (MFCCs) are used as inputs of the networks.

We decided to use Keras, a Python library for neural network implementation, which is capable to run on top of software as Theano and Tensorflow. We also used Matlab to evaluate and represent different outputs generated.

Data used, both train and evaluation, have been taken from the database OpenSAT (Open Speech Analytic Technologies) developed by the National Institute of Standards and Technology (NIST). In this database we can find more databases from which we can extract sound signals and data. For this Bachelor Thesis we have used VAST (Video Annotation for Speech Technologies) database, audios from a real fire response from PSC (Public Safety Communications) and recorded conversational audios from IARPA Babel (Intelligence Advanced Research Projects Activity).

To evaluate the way in which one system is better than the other one, we have decided to use accuracy and Equal Error Rate (EER) metrics calculated using the evaluation data. This metrics are used to compare each model's performance.

To conclude, we have developed a LSTM that returns a better response than the DNN, as we thought it would happen, due to LSTM performing better in tasks using temporal signals. We have also demonstrate that this Voice Activity Detector with data we chose returns a better classification using a LSTM with less context information as input. We have seen as well that in a DNN model, a simple one works better.

Keywords (inglés)

Voice Activity Detection, Neural Network, DNN, LSTM, *Accuracy*, EER

Agradecimientos

En primer lugar, quiero dar las gracias a mi tutora Alicia por aguantar mis incansables correos con preguntas de última hora durante el último mes. Gracias a ella he conseguido sacar adelante en el último momento este Trabajo Fin de Grado en el que me encontraba tan perdida. En segundo lugar, gracias a mi "segundo tutor" por soportar mis continuas quejas y constantes preocupaciones por si iba a terminar a tiempo. Gracias Richi, especialmente por guiarme y animarme.

Aunque cada uno haya estado en su mundo este año, gracias a mis telecos, en especial a Julito, Ana y Clau. A mis hermanos, a pesar de ser uno de los peores años por lo que hayamos pasado, somos demasiado cabezotas como para no superarlo así que os dedico estos años de carrera en los que me habéis aguantado. A las Cepeda, por hacerme reír estos tres últimos años y animarme a base de chuches.

Y, por último, aunque no me hayas aguantado todos los años de carrera, has tenido suficiente con estos últimos tres (aunque probablemente te toque hacerlo muchos años más). Gracias por hacer todo más fácil.

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1 Introducción

1.1 Motivación

En los últimos años, el uso de la tecnología ha aumentado rápidamente y seguirá aumentando, sobre todo, en la forma en la que cualquier dato es procesado por máquinas diariamente en nuestras vidas. Llegará el momento en el que estas máquinas esperarán a que los humanos introduzcamos datos o información, en lugar de esperar nosotros a que las máquinas respondan. Esto último se debe a que, poco a poco, esta tecnología va ganando poder y rapidez en el procesamiento gracias a las mejoras que van surgiendo. Hoy en día, las máquinas son capaces de aprender a partir de los datos que se les proporcionan; pueden incluso recibir y entender órdenes que reciben. Estos datos pueden llegar a ser datos complejos como la propia voz.

Teniendo en cuenta que el ámbito de este Trabajo Fin de Grado es el de las Telecomunicaciones, lo primero que se viene a la cabeza al pensar en "comunicación" es el intercambio de palabras, de sonidos, de voz. Por ello, una de estas mejoras tecnológicas interesante de estudiar es la relacionada con la detección, el reconocimiento y la clasificación de voz o habla, la cual está atrayendo la atención de forma notable en la actualidad. Ejemplos conocidos son herramientas como los asistentes de voz de Apple, Siri, o Windows, Cortana. A pesar de las diferencias entre las distintas tareas relacionadas con el procesamiento de la señal de voz, dependiendo de la información que se quiera extraer de la misma, cualquier sistema relacionado con la señal de voz necesita complejos sistemas para modelar la información contenida en la señal. Para su desarrollo, se necesitan procesadores rápidos y grandes requerimientos de memoria.

Este Trabajo Fin de Grado se centrará en la Detección de Actividad de Voz (en inglés, *Voice Activity Detection*, VAD), es decir, la detección de forma automática de los segmentos de una señal de audio (grabación) en los que se presenta o no habla (detección de voz y silencio por tramas). Este módulo, es típicamente un paso previo al procesamiento de la señal de voz en cualquier sistema de reconocimiento, de codificación o de mejora de voz. Por tanto, es una interesante tarea a conocer y estudiar en estos ámbitos de la tecnología y las comunicaciones.

Como en muchos otros campos, las redes neuronales tienen una amplia historia en el reconocimiento de voz, normalmente en combinación con los Modelos de Mezclas de Gaussianas (*Gaussian Mixture Models*, GMMs). Debido a las drásticas mejoras en modelado acústico junto con redes profundas pre-alimentadas (*feed-forward*), estas últimas han ganado importancia en muchas tareas relacionadas. La mayoría de sistemas en el estado del arte en la actualidad para la tarea de VAD están basados en redes neuronales profundas (*Deep Neural Networks*, DNNs), que son capaces de detectar las tramas de voz y no voz a partir de distintas características acústicas de entrada tales como los Coeficientes Cepstrales de Frecuencias Mel (*Mel Frequency Cepstral Coefficients*, MFCCs).

Además, debido a que el habla es un proceso dinámico (la señal de voz es una señal temporal), las redes neuronales recurrentes (*Recurrent Neural Networks*, RNNs) son ideales para modelar dicha señal. En especial, las *Long Short-Term Memory*, LSTM, las cuales tienen en cuenta el contexto temporal para modelar la información en el tiempo actual. Probablemente han aparecido ya o van a ir apareciendo distintos tipos de redes

neuronales que, en solitario o combinadas y con otros tipos de entrenamiento y parámetros, vayan consiguiendo mejores resultados en la tarea de VAD.

En este caso de estudio, resulta interesante implementar y analizar la DNN descrita anteriormente para analizar distintas señales de audio y conseguir una predicción lo más veraz posible de habla o no habla. Además, una comparación con otro tipo, como la LSTM, proporcionaría una base útil a la hora de elegir entre varios sistemas distintos o hacer una evaluación de resultados.

1.2 Objetivos

El objetivo principal de este Trabajo Fin de Grado es desarrollar un sistema de Detección de Actividad de Voz (VAD). Este sistema consistirá en el entrenamiento de redes neuronales de varias capas con características de entrada Coeficientes Cepstrales en las frecuencias de Mel (*Mel Frequency Cepstral Coefficients*, MFCCs). Por tanto, se busca implementar el mejor sistema con la arquitectura necesaria para obtener los mejores resultados.

Para demostrar qué arquitectura ofrece mejores resultados en la detección de voz, se realizarán varios experimentos en cada entrenamiento para su posterior comparación empírica. Esta evaluación, como ya se ha dicho previamente en la motivación, se realizará sobre dos casos de estudio: *Deep Neural Networks* (DNNs) y *Long Short-Term Memory Neural Networks* (LSTMs).

1.3 Organización de la memoria

La memoria presenta la siguiente distribución: Resumen/*Abstract*, Palabras clave/*Key words*, Agradecimientos, Índices, Trabajo y Referencias. La parte que detallar es la de "Trabajo", la cual se desarrolla de la siguiente forma:

- *Introducción:* Se ha expuesto la motivación del Trabajo Fin de Grado, los objetivos que este presenta y este mismo apartado, *Organización de la memoria*.
- *Estado del arte:* Se desarrolla la situación actual y fundamentos de la Detección de Actividad de Voz, así como de las Redes Neuronales Profundas y Recurrentes.
- *Entorno experimental:* Se detallan los datos, técnicas y métodos utilizados para el desarrollo del sistema y su posterior evaluación.
- *Diseño y desarrollo:* Se plantea el trabajo dividido en varias fases, explicando en qué consiste cada una.
- *Pruebas y resultados:* Se explican qué parámetros se han variado en cada experimentación y se analizan los resultados obtenidos procediendo a la comparación.
- Conclusiones y trabajo futuro: Se exponen la conclusión y las distintas líneas de trabajo en las que se podría actuar para mejorar resultados.
- *Referencias*

2 Estado del arte

2.1 Sistemas de Detección de Actividad de Voz

2.1.1 Introducción

El término Detección de Actividad de Voz (*Voice Activity Detection*, VAD) se refiere a una clase de métodos de procesamiento de la señal que detectan si fragmentos de audio hablado contienen datos de voz o no **[1]**.

Como se ha dicho en la *Motivación*, un sistema VAD tiene un papel esencial en aplicaciones de procesado de voz tales como mejora, codificación y reconocimiento **[2]**, ya que es interesante poder hacer una previa clasificación de estos datos en hablados o no hablados para su posterior procesamiento.

Un proceso de comunicación como los mencionados con un sistema VAD integrado presenta varias mejoras entre las que destacan la mejora de la capacidad del canal, la reducción de interferencias de canales adyacentes y de consumición de potencia en los dispositivos. En ambientes no estacionarios donde la señal de voz puede verse afectada por interferencias o ruido, el sistema VAD puede llegar a utilizarse para aprender características de estas señales interferentes y estimar su espectro, de tal forma que se podría llegar a suavizar o incluso eliminar su impacto. También, si lo requiere el sistema se puede utilizar un detector de actividad de voz para identificar los segmentos de silencio (o no voz) y descartarlos.

El funcionamiento de un detector de actividad de voz suele basarse siempre en tomar decisiones basadas en características particulares de la señal. Este caso de estudio tendrá en cuenta solo los sistemas VAD para datos de audio.

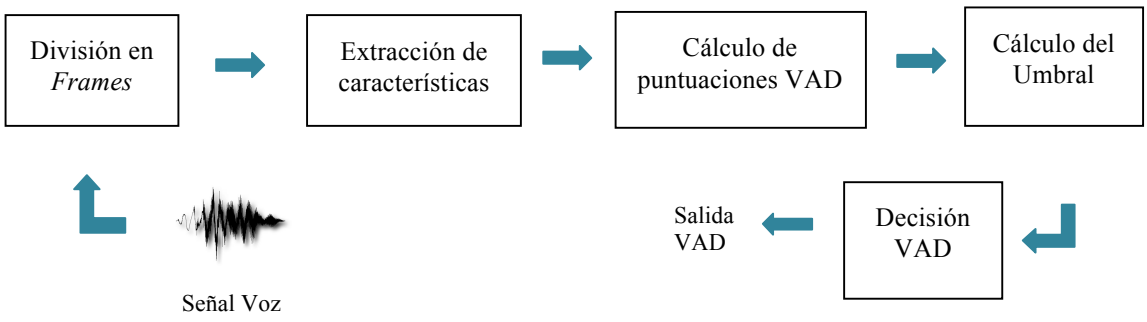


Figura 2-1: Diagrama de bloques del proceso de Detección de Actividad de Voz

Como indica el diagrama de la **Figura 2-1**, la señal de entrada pasa por una primera etapa de división en *frames* de un tamaño específico para hacer un mejor procesamiento de la señal (a corto plazo, en el orden de decenas de milisegundos); la segunda etapa es la de la extracción de características en la que se representa cada *frame* con un vector de características, la cual se tratará en el apartado *Extracción de Características*; la tercera sería la de cálculo de puntuaciones del sistema VAD para cada *frame*; la cuarta etapa es

una decisión por parte del detector a través de un cálculo del umbral para la posterior clasificación; y, por último, se representa la salida con la decisión generada.

2.1.2 Extracción de Características

A lo largo de los años, han ido surgiendo distintos enfoques en cuanto a la detección de voz en señales. En **[3]** se hace una breve introducción a este tema. La principal diferencia entre ellos es la de las características usadas para representar la señal de audio.

Los primeros algoritmos se basaban en la extracción de características en el dominio del tiempo como energía a corto plazo (*Short-Time Energy*, STE), tasa de cruces por cero (*Short-Time Zero Crossing Rate*, STZCR) y magnitud media a corto plazo (*Short-Time Average Magnitude,* STAM). Sin embargo, muchos de estos algoritmos, como STE y STZCR, han ido mostrando problemas por una baja relación señal-ruido (*Signal to Noise Ratio*, SNR), sobre todo cuando el ruido es no estacionario.

A medida que han ido avanzando los experimentos y resultados, se han ido desarrollando características más robustas, como las basadas en la función de autocorrelación, los coeficientes cepstrales, transformación Wavelet, medida de periodicidad y modelos estadísticos.

Dependiendo del tipo de aplicación posterior, ya sea reconocimiento **[4],** codificación **[5]** o compresión **[6]**, se puede diseñar un tipo de VAD específico como figura en las referencias respectivamente.

2.1.2.1 Coeficientes Cepstrales en las Frecuencias de Mel (MFCC)

Una de las representaciones más utilizadas en los sistemas de procesamiento de voz, y las que en este caso se han elegido como características de entrada a los sistemas desarrollados en este trabajo son los Coeficientes Cepstrales en las Frecuencias de Mel (*Mel Frequency Cepstral Coefficients*, MFCCs) **[7].** Dichos coeficientes se basan en la percepción del oído humano para la representación de la voz.

Estas características han sido las dominantes durante un largo periodo de tiempo para reconocimiento de voz debido a su habilidad para representar el espectro de la amplitud de forma compacta, además de su robustez frente al ruido. Una completa descripción del proceso de cálculo de estas características se da en **[8].** La **Figura 2-2** muestra un diagrama del proceso a grandes rasgos.

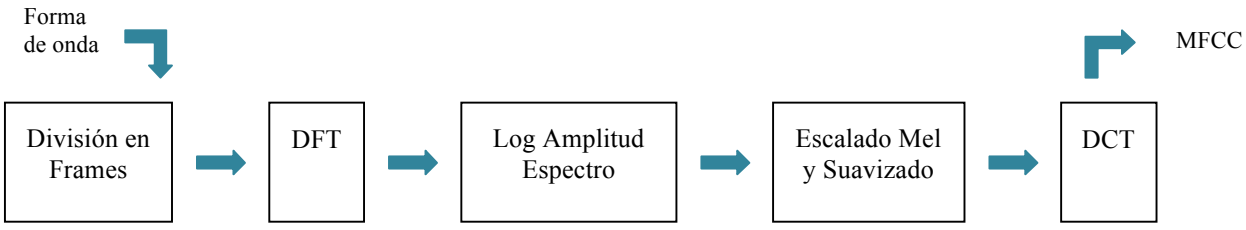


Figura 2-2: Diagrama de bloques del proceso de Extracción de Características

2.1.3 Modelos Tradicionales de VAD

Uno de los modelos acústicos de procesado de señal de voz para aprendizaje automático ha sido siempre el Modelo de Mezclas de Gaussianas *(Gaussian Mixture Models*, GMM), modelo probabilístico que asume que todos los datos son generados mediante una mezcla finita de distribuciones Gaussianas con parámetros desconocidos.

En el caso particular de VAD, un GMM se puede utilizar para modelar el espacio de características (MFCCs, por ejemplo). Algunas aproximaciones utilizan este modelo Gaussiano entrenado (con dos componentes Gaussianas) para la probabilidad de que las características provenientes de un fragmento de señal de audio sea voz o no voz.

Se trata de un modelo bastante flexible capaz de modelar un amplio rango de distribuciones. Sin embargo, tiene la principal desventaja de que se produzca *overfitting*, es decir, el sistema se ajustaría demasiado a unos datos de entrenamiento y no sería capaz de generalizar a datos no vistos durante este entrenamiento. Por tanto, se requiere gran cantidad de datos distintos para conseguir un entrenamiento efectivo.

Otra aproximación que destacar es el modelo de Máquinas de Vectores de Soporte (*Support Vector Machines*, SVMs), el cual, dado un conjunto de datos de entrenamiento, construye un modelo que asigna nuevos ejemplos a una categoría, generando un hiper-plano que separa de forma óptima una clase de otra buscando que la distancia entre clases sea máxima.

En **[9]** se demuestra la efectividad de un modelo SVM para una detección robusta de detección de voz comparándolo con otros estándares como ITU-T G.729 y otros sistemas de VAD que definen su decisión basándose en medias calculadas de características. Se define una decisión no lineal en base a un vector de características que incluye información contextual.

Existen otros muchos modelos utilizados para la Detección de Actividad de Voz. En **[10]** se hace una comparativa de los dos anteriores y el modelo de Árbol de Decisión (*Decision Tree,* DT) para esta tarea. Concluye que el mejor clasificador es el DT en cuanto a precisión y coste computacional. Este modelo mapea observaciones sobre una muestra a conclusiones sobre el objetivo de esta muestra, es decir, se predice el valor de una variable en función de diversas variables.

2.2 Redes Neuronales

No hay "máquina" más compleja de entender para el ser humano que su propio cerebro, compuesto por casi unos 100 mil millones de neuronas y con más de 500 mil de millones conexiones entre ellas. Es por esto por lo que, para intentar avanzar en el entendimiento de su funcionamiento, se intenta reproducir o emular de cierta forma. Tareas diarias como el simple hecho de reconocer un rostro entre varios son complejas a la hora de reproducirlas en una máquina. Estos reconocimientos y clasificaciones de patrones requieren tiempo y esfuerzo comparado con el proceso en el cerebro humano.

Gracias a este interés en desarrollar un comportamiento parecido al del cerebro, un aprendizaje de la máquina surge la Red Neuronal Artificial (*Artificial Neural Network*, ANN). Por tanto, una red neuronal es un modelo matemático inspirado en este comportamiento y estructura del cerebro, el cual puede llegar a realizar complejos cálculos

para un aprendizaje automático a partir de los datos. La **Figura 2-3** muestra una comparación entre las neuronas biológicas y las artificiales.

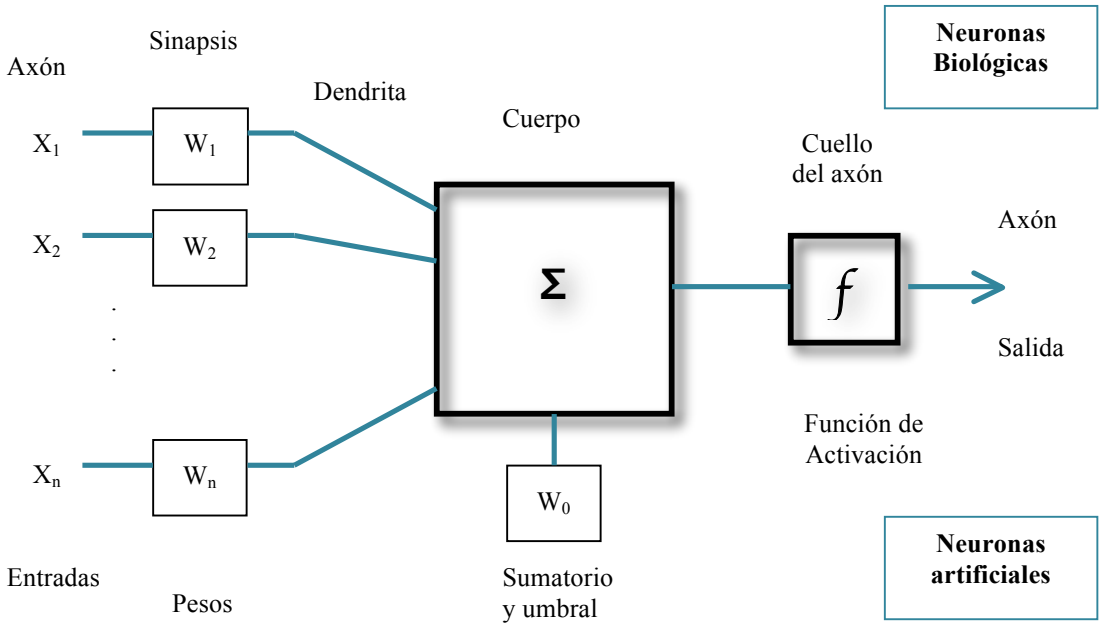


Figura 2-3: Diagrama de comparación entre neuronas artificiales y biológicas

Las características principales de las redes neuronales artificiales podrían organizarse en los tres puntos siguientes **[11]**:

- *Autoorganización:* para ofrecer un procesado robusto.
- *Procesado no lineal:* para aproximar funciones y para la clasificación de patrones aumenta su capacidad y su inmunidad frente al ruido e interferencias.
- *Procesado paralelo:* presenta nodos de procesado con alto nivel de interconectividad.

Como se representa en la **Figura 2-4**, las redes neuronales consisten en un par de capas de entrada y salida, a las cuales se les añade entre medias una o varias capas ocultas. Cada capa se compone de unidades ocultas (neuronas) que transforman mediante ponderaciones junto con el uso de cierto tipo de función de activación (no lineal), la entrada en una salida.

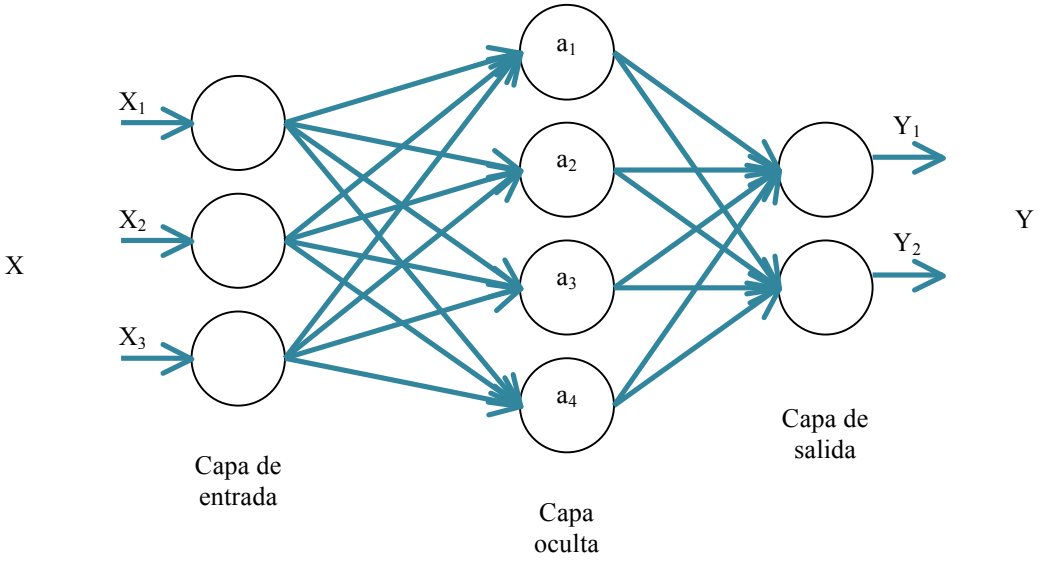


Figura 2-4: Esquema de una red neuronal

Para entender bien el funcionamiento, en la **Figura 2-5** se muestra una neurona, la unidad básica de la red neuronal. Cada entrada x_1 y x_2 está conectada con la unidad oculta mediante su correspondiente peso w_n, el cual se estima de forma automática mediante el proceso de entrenamiento de la red neuronal. Este peso asociado se irá modificando a lo largo del proceso de aprendizaje en toda la red neuronal, tratando de minimizar una función de coste dada. La salida $h_w(x)$, corresponderá al resultado final aplicándole la función de activación $g(\cdot)$, la cual se tratará más adelante. Cada unidad aplica una función que proviene de la suma de los inputs ponderados mediante los pesos, como muestra la fórmula de la **Figura 2-6**.

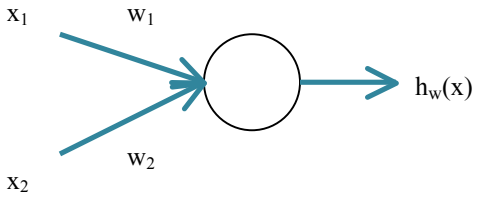


Figura 2-5: Esquema de una neurona

$$h_w(x) = g(w_1 * x_1 + w_2 * x_2)$$

Figura 2-6: Cálculo de la salida de la neurona

La estructura vista hasta ahora es bastante simple. Se pueden distinguir varias topologías cuyo uso dependerá de las necesidades del diseñador de la red. Más adelante se tratarán los dos tipos de arquitecturas que se utilizan en este Trabajo Fin de Grado.

2.2.1 Función de Activación y Proceso de Aprendizaje

Como se ha dicho anteriormente, una neurona o unidad de las capas ocultas presenta una transformación que generará un resultado final para el entrenamiento de la red. Esta transformación no lineal $g(\cdot)$ aplicada sobre la operación lineal de la **Figura 2-6** es la función de activación, la cual calcula el estado de actividad de una neurona; transformando la entrada en un valor de activación, cuyo rango normalmente va de [0 a 1] o de [-1 a 1].

Existen diversas funciones de activación para los nodos de la red. En la **Tabla 2-1** se destacan las más usuales.

	Función	**Rango**
Identidad	$g(x_i) = x_i$	$[-\infty, +\infty]$
Escalón	$g(x_i) = sign(x_i)$	[-1, +1]
Sigmoidea	$g(x_i) = 1/(1+e^{-xi})$	[0, +1]
Gaussiana	$g(x_i) = Ae^{-B(xi\hat{}2)}$	[0, +1]
Sinusoidal	$g(x_i) = Asen(\omega x_i + \varphi)$	[-1, +1]
Softmax	$g(x_i) = e^{xi}/(\Sigma_i e^{xi})$	[0, +1]
ReLU	$g(x_i) = max(0, x_i)$	$[0, +\infty]$

Tabla 2-1: Funciones de Activación

En el apartado de *Pruebas y Resultados* se explicará cuál de ellas se ha decidido utilizar para los modelos de redes neuronales generados para la experimentación del Detector de Actividad de Voz. Sin embargo, se quieren destacar tres que resultan interesantes para este Trabajo Fin de Grado.

- *Sigmoid:* función no lineal que se usa sobretodo en modelos en los que se hace una predicción de probabilidad en la salida ya que da como salida un valor en el rango [0,1]. Es muy útil en clasificaciones de dos clases, es decir, clasificaciones binarias. Sin embargo, puede causar que una ANN se quede parada en el tiempo de entrenamiento **[12].**
- *Rectifier Lineal Unit (ReLU):* función no lineal usada actualmente en casi todos los problemas de redes neuronales convolucionales. Tiene una desventaja en su habilidad para entrenar datos correctamente, ya que considera nulos todos los valores negativos de entrada. A pesar de ello, evita que los pesos se queden parados durante el entrenamiento.
- *Softmax:* función no lineal similar a la primera descrita pero más indicada para problemas de clasificación en varias clases.

En la **Figura 2-7** se representan simplemente las dos primeras funciones ya que la *softmax* no es representable en dos dimensiones debido a su característica de clasificación multi-clase.

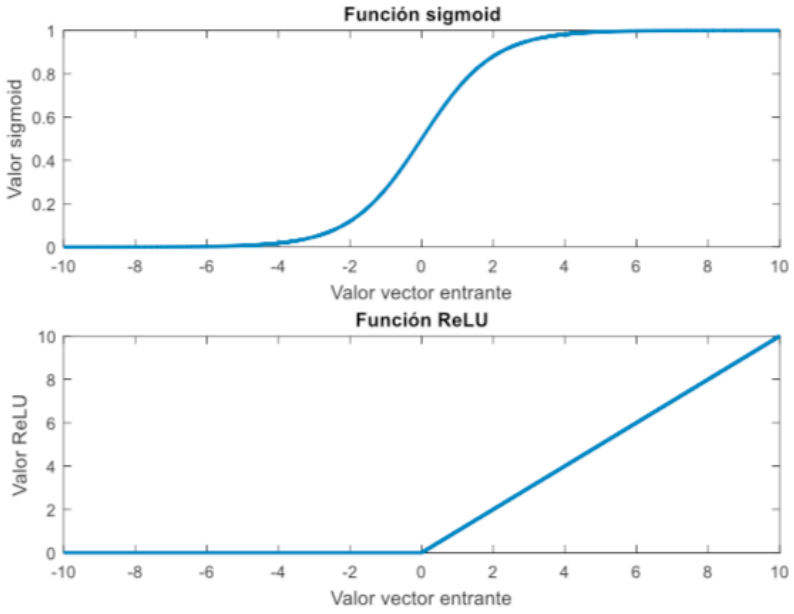


Figura 2-7: Gráficas de las Funciones de Activación [12]

El entrenamiento de la red o aprendizaje tiene como objetivo ajustar (aprender) unos valores para los parámetros que, transformando la entrada, la mapea a una salida y consigue minimizar una función de coste. Esta función de coste mide la adaptación de la red al objetivo, es decir, indica matemáticamente la relación entre el objetivo y el resultado final del aprendizaje. Por tanto, se busca minimizar todo lo posible este resultado. El comportamiento de esta relación variará dependiendo del parámetro de la tasa de aprendizaje, la cual permite una mayor o menor actualización de los pesos calculados.

Se van a destacar dos tipos de función de coste, de las cuales se elegirá una que se indicará más adelante:

- *Entropía cruzada (Cross-Entropy):* Usada en problemas de clasificación, compara de alguna manera las probabilidades de salida (0, 1). Normalmente la distribución calculada se expresa en términos *one-hot*. Siendo *p(x)* la probabilidad deseada y *q(x)* la obtenida:

$$H(p,q) = -\sum_{x} p(x)\,\log q(x).$$

Figura 2-8: Cálculo de la Entropía Cruzada

- *Error cuadrático medio (Mean Squared Error, MSE):* Útil sobre todo en problemas de regresión. Mide la diferencia en mínimos cuadrados entre los datos y las predicciones, por tanto, representa la cercanía de la hipótesis a los datos reales. Siendo m, el número

de muestras se calcula la distancia entre ambos puntos, $h_\Theta(x)$ siendo la hipótesis e y siendo el real:

$$\frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$$

Figura 2-9: Cálculo del Error Cuadrático Medio

El minimizado de esta función se consigue mediante el descenso por gradiente o algoritmos similares basados en el *Back-Propagation* **[13]**, algoritmo de ajuste de los pesos calculados para las neuronas. Para ello, se necesita que la red compute el error derivado de la variación de los pesos para conseguir un error menor en cada iteración hasta que la red converja a un valor deseado.

2.2.2 Redes Neuronales Profundas

El aprendizaje profundo es una parte del aprendizaje automático que presenta como principal diferencia que, en lugar de aplicar algoritmos específicos para una tarea determinada, se basa en un aprendizaje de las representaciones de los datos.

Las Redes Neuronales Profundas *(Deep Neural Networks*, DNNs) son estructuras que realizan un aprendizaje de este tipo. Son redes neuronales artificiales con varias capas ocultas entre la de entrada y la de salida, que actúan aplicando una propagación *feed-forward,* es decir, los datos fluyen de la primera a la última capa (no hay "*loops*").

Hay muchas investigaciones acerca de por qué se usan redes neuronales de más de una capa en la mayoría de los casos de estudio. Sin embargo, no se ha llegado a una razón concluyente. Se apunta a que cada capa aprende una representación de la entrada a distintos niveles de abstracción o a que también tienen menos parámetros que una red de una sola capa con muchas más neuronas. A pesar de tener una cierta preferencia hacia las múltiples capas, se intenta demostrar también que se podría llegar al mismo resultado con una capa oculta única **[14]**.

2.2.3 Redes Neuronales Recurrentes

Al igual que las DNNs, las Redes Neuronales Recurrentes (*Recurrent Neural Network*, RNNs) son un tipo de redes neuronales artificiales. En este caso, como entrada no solamente tienen en cuenta la entrada la actual, si no que toman cierta cantidad de datos que han recibido anteriormente, es decir, tienen "memoria". En la **Figura 2-10** se observa un esquema de esta red.

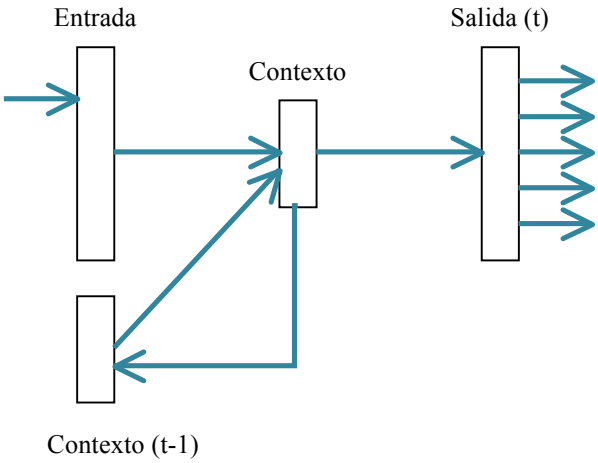


Figura 2-10: Esquema de una RNN

Como diferencia adicional frente a las DNNs relacionada con la "memoria", las RNNs fluyen hacia atrás para conectar con decisiones pasadas, tomando sus propias salidas como entradas a la red. Gracias a esto, se consigue tener como entrada cierta información de la secuencia (contexto).

Estas redes son más eficaces para resolver problemas que presentan no-linealidades temporales significativas, es decir, para aplicaciones cambiantes en el tiempo.

2.2.3.1 LSTM

Las redes neuronales recurrentes LSTM (*Long Short-Term Memory*) **[15]** son una variación de las anteriores descritas que corrigen el problema del *vanishing gradient,* en el cual, en algunos casos, el gradiente de la función de error disminuye de tal forma que el peso calculado no varía en su valor. Esto puede producir que la red deje de aprender, por tanto, las LSTMs ayudan a preservar el error manteniéndolo más constante a través de su capacidad de aprender dependencias a largo plazo (de ahí su nombre).

En la **Figura 2-11** y en su ampliación, la **Figura 2-12,** se observa que en las RNNs el modelo es el de una simple capa de una tangente, por ejemplo. Mientras tanto, en las LSTMs este módulo tiene una estructura algo más compleja: presenta cuatro puertas o unidades que interaccionan entre sí. En el diagrama cada línea lleva un vector entero desde la salida de un nodo a la entrada de otro. Los círculos azules de la **Figura 2-12** representan operaciones punto a punto y las cajas son las capas de aprendizaje, como se ha dicho antes.

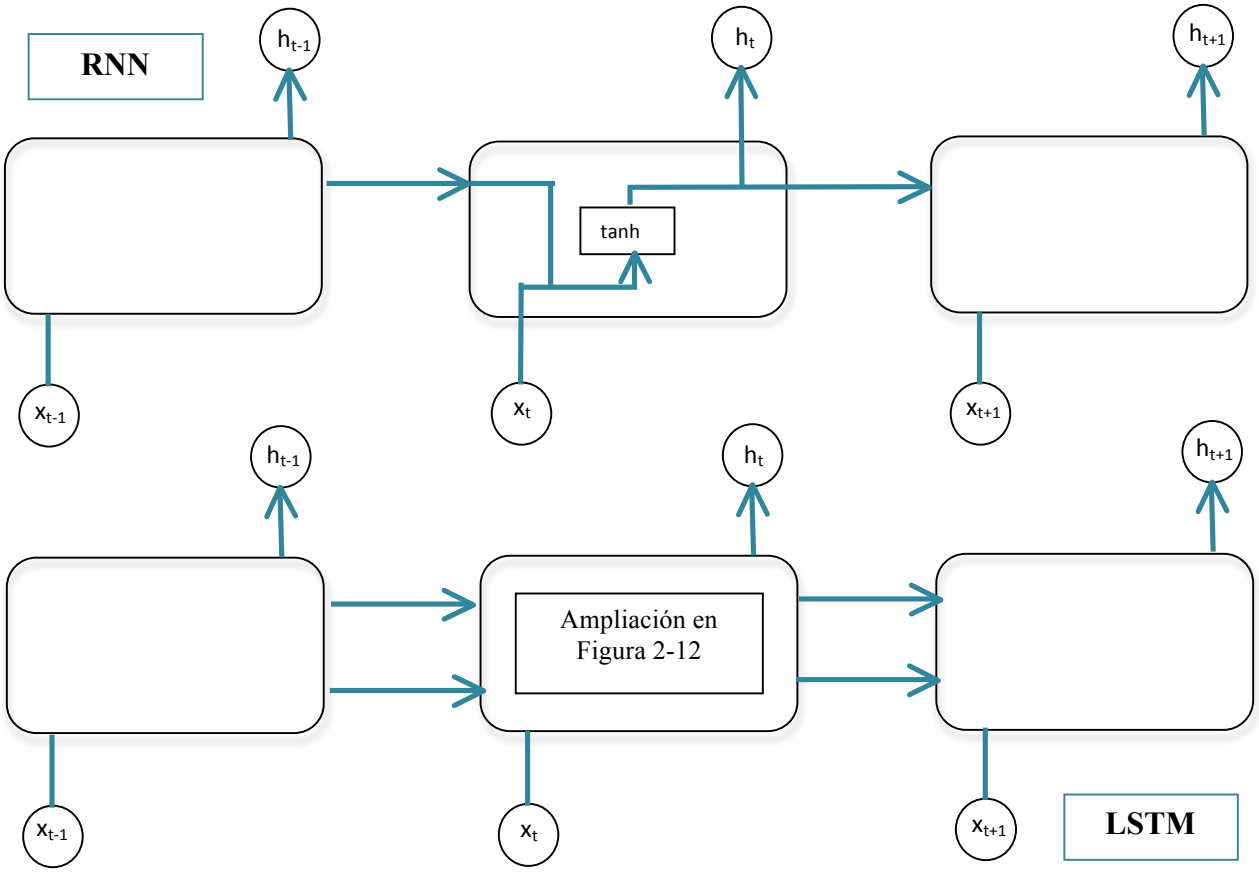


Figura 2-11: Comparación RNN-LSTM [16]

Este tipo de redes contienen información fuera de la propagación en una *gated cell,* de tal forma que pueden guardar, escribir o leer datos de ella cuando esta celda se lo permita. En este Trabajo Fin de Grado no se describe en profundidad el funcionamiento de esta red, pero un ejemplo de celda sería el de la **Figura 2-12,** con sus respectivos componentes.

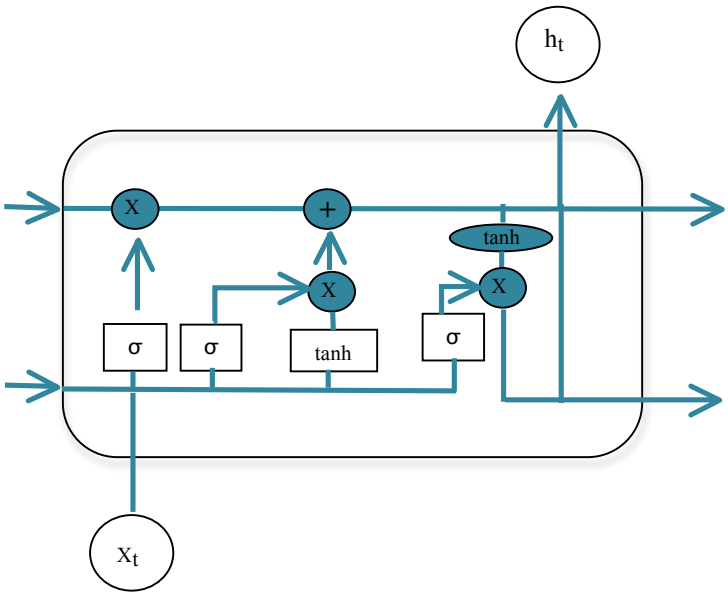


Figura 2-12: Celda LSTM [16]

La explicación del esquema a grandes rasgos es el siguiente:

- La línea horizontal superior representa el estado de la celda, en la cual se producen operaciones lineales.
- Cada uno de los bloques amarillos es una capa de la red neuronal. Tres sigmoidales (σ) y una tangencial (tanh).
- Las puertas (*gates*) dejan pasar o no información, lo cual lo decide la capa sigmoidal. La primera decide qué información se deshecha (*forget gate layer*); la segunda decide qué información nueva se va a guardar en el estado de la celda, interviniendo la segunda capa (*input gate layer*) y la capa tangencial, la cual crea un nuevo vector de valores candidatos a añadir; y, por último, se decide qué se va a sacar por la salida. Para esto último, se usa la capa que decide qué partes del estado van a mostrarse como salida y se pasan por una tanh, para situar los valores en el rango [-1, 1].

2.2.4 Detección de Actividad de Voz mediante Redes Neuronales

Se han encontrado diversos estudios acerca del uso de distintos tipos de Redes Neuronales Artificiales para la tarea de Detección de Actividad de Voz en un amplio rango de situaciones. Se ha ido buscando en función del hilo que se va a seguir en el Trabajo Fin de Grado.

Para comenzar se ha buscado cómo se pueden extraer las características de entrada a la red neuronal. Aunque en este trabajo no se implementa, es interesante ver cómo se podría haber hecho haciendo uso de NN, ya que es una de sus muchas aplicaciones.

En **[17]** y **[18]** se procede a esta extracción. En el primer *paper* se hace mediante Redes Neuronales Convolucionales (*Convolutional Neural Network*, CNN) con el fin de utilizar estas características a un VAD; en el segundo, en cambio, se extraen para un proceso de reconocimiento de idioma. Elegir las características apropiadas va relacionado con la elección de la arquitectura usada, por tanto, es una decisión importante para el entrenamiento de la red.

A continuación, se han encontrado diversos estudios sobre redes neuronales en VAD. Sin embargo, casi todos los casos tratan el tema de VAD con ruido añadido. En **[19]** se ha entrenado una red neuronal artificial de forma óptima para proveer un modelo de detección de actividad de voz en ambientes hostiles. Se ha optimizado el modelo afectado por distintos tipos de sonido (bases de datos TIMIT y SNOW) observando cómo los algoritmos que ya existían para VAD (ITU-T Rec. G.729 Annex B, ETSI AFE ES 202 050) trabajaban en esta situación y mejorándolo. Como resultado muestran que la red neuronal con características de entrada MFCC propuesta proporciona valores mucho mejores ante condiciones con interferencias, además de conseguir una clasificación mucho más exacta.

[20] propone usar una DNN para aprender la relación entre características de voz con ruido y la correcta decisión en VAD. Explican que en casi todos los casos de estudio se necesita un algoritmo causal, es decir, trabajar a tiempo real y utilizando solo fragmentos de audio

presentes y pasados. Obtienen una importante reducción del error de clasificación para los distintos tipos de ruido, aunque para el caso de ruido "invisible" se esperaba un resultado mejor. Proponen como trabajo futuro experimentar con otras redes como las recurrentes para preservar el estado y reducir el tamaño del vector de entrada.

Esta última propuesta de trabajo futuro ha hecho que se busquen casos en los que la detección de actividad de voz se modele con una red recurrente. En el caso de **[21]** se presenta una RNN multicapa donde los nodos computan polinomios cuadráticos para VAD. La comparan con modelos de mezclas gaussianas junto con una máquina de estados para suavizado temporal, consiguiendo un 26% de reducción en falsas alarmas y reduciendo la computación un 17%.

En los dos siguientes casos se entrenan redes neuronales para VAD para una posterior aplicación en alguna tarea específica.

Por ejemplo, en **[22]** se entrena una LSTM siguiendo el estándar RASTA-PLP de características *front-end* para aproximarlo a un escenario de la vida real con grandes cantidades de ruido. Se testea datos de cuatro películas de Hollywood y en señales de habla de las bases de datos TIMIT y *Buckeye corpora*. Se obtiene un *Equal Error Rate* de 33,2% para las películas y 9,6% para los otros datos con un SNR de 0dB, mejorando bastante los resultados de los VAD tradicionales que describen anteriormente en el estado del arte.

Otro caso de estudio es **[23]**, en el cual comenta que los métodos normalmente usados como modelos de mezclas gaussianas no trabajan bien en ambientes como *Youtube* donde los vídeos están expuestos a un rango enorme de condiciones ambientales. Proponen aprender características robustas durante el entrenamiento usando DNNs. Demuestran que una DNN compuesta de una entrada de MFCCs consigue ratios de error mucho menores (19,6%) que en sistemas GMMs (40%).

Por tanto, se puede concluir que las redes neuronales aplicadas a la tarea de VAD se utilizan en la mayoría de los casos para limpiar la señal de audio, es decir, aprenden las condiciones ruidosas o de interferencias para poder obtener una señal lo más veraz a la real. Además, se ha visto que se aplican también a la extracción de características de entrada.

3 Entorno Experimental

3.1 Herramientas

En esta sección del TFG se explicarán brevemente las tecnologías utilizadas para la realización del diseño y los experimentos llevados a cabo, así como el entorno utilizado para comparar los resultados obtenidos. Además, se hará una breve descripción de las bases de datos utilizadas para el entrenamiento y para las pruebas finales.

3.1.1 Lenguajes de Programación

Como lenguaje de programación para la implementación de los scripts se ha decidido usar Python, lenguaje con licencia de código abierto y multiplataforma. Básicamente, su uso se debe a las librerías que proporciona para la tarea de *machine learning*, las cuales se describirán más adelante.

Gracias a que en los últimos años se ha utilizado cada vez más este entorno para el aprendizaje automático, cualquier problema que se tenga es fácilmente solucionable ya que se van actualizando soluciones y mejoras a través de páginas de soporte (*StackOverFlow*) de personas interesadas en este ámbito.

En **[24]** se han comparado distintos lenguajes como Python, C y R para ver su rendimiento y sencillez de código a la hora de implementar aplicaciones de aprendizaje automático y, aunque no esté muy por encima, resulta fácil afirmar que va a incrementar su uso el primero en esta área.

3.1.2 Librerías

Las librerías utilizadas para este proyecto se encuentran en repositorios de libre acceso y se van actualizando por desarrolladores continuamente.

Principalmente, el código se implementó haciendo uso de la librería Keras **[25],** la cual es capaz de ejecutarse por encima de TensorFlow **[26],** Theano **[27]** y Microsoft Cognitive Toolkit, entre otros. Está diseñada para habilitar la experimentación rápida con redes neuronales profundas independientemente del *backend* elegido, lo cual es muy beneficioso para este trabajo.

El script base del que se partió estaba implementado usando la librería de Theano, que permite definir y evaluar expresiones matemáticas multidimensionales con métodos mucho más complejos y menos intuitivos que Keras. Por ello, se volvió a implementar este código, pero esta vez haciendo uso de los métodos proporcionados por Keras. Por otra parte, se decidió en un principio usar esta librería como *backend* de Keras, pero debido a que a finales de 2017 se decidió dejar de darle soporte, se cambió a Tensorflow.

Keras contiene numerosas implementaciones de bloques necesarios para la construcción de una red neuronal tales como capas, funciones de activación y optimizadores, interesantes a la hora de implementar los dos modelos necesarios para el estudio. Además, permite realizar el entrenamiento de los modelos de aprendizaje automático en Unidades de Procesamiento Gráfico (*Graphics Processing Units*, GPUs).

En cuanto a Tensorflow, es una librería de código abierto que satisface las implementaciones de sistemas capaces de generar y entrenar redes neuronales para obtener patrones y correlaciones en los resultados. En cuanto a la ejecución física, identifica las GPUs del sistema e intenta obtener el máximo rendimiento en ellas, lo cual es distinto en Theano.

3.1.3 Matlab

Matlab (*MATrix LABoratory*) **[28]** es una herramienta de software matemático que proporciona un entorno de desarrollo con lenguaje propio para manipulación de matrices, representación de funciones, visualización de datos e implementación de algoritmos, entre otras prestaciones. Se ha hecho uso de esta herramienta sobre todo para la representación de resultados obtenidos o de formas de onda de los sonidos proporcionados y, además, se ha utilizado para ejecutar algoritmos que calculan errores para la posterior comparación de resultados.

3.1.4 GPUs

La GPU es un coprocesador capaz de renderizar imágenes y procesar audios mucho más rápidamente que una CPU (*Central Processing Unit*) debido a su arquitectura de procesamiento paralelo, la cual permite bastantes cálculos al mismo tiempo. El uso de las unidades de procesamiento gráfico (*Graphics Processing Unit*, GPUs*)* para ejecutar los scripts ha sido clave debido a la cantidad de datos y el entrenamiento de la red, dando una diferencia de muchas horas en tiempo de computación gracias a su potencia. Para explotar al máximo el uso de la GPU se usa CUDA junto con la librería cuDNN **[29]**, la cual beneficia el tiempo de cómputo de algoritmos para el entrenamiento mediante redes neuronales.

3.1.5 Bases de datos

Como se mencionó en el *Resumen*, los datos se han obtenido de la base de datos OpenSAT (*Open Speech Analytic Technologies*) **[30]**, OpenSAT es una de evaluación de análisis de tecnologías referidas al habla que ha organizado el Instituto Nacional de Estándares y Tecnología (*National Institute of Standards and Technology,* NIST). Incluye tres tareas principales: VAD, Búsqueda de palabra clave (*Key Word Search*, KWS) y Reconocimiento automático del habla (*Automatic Speech Recognition,* ASAR) y evalúa estos tres procesos en base a: lenguajes con pocos recursos, habla de vídeos y comunicaciones de seguridad pública.

Particularmente se compone de datos de la base de datos VAST (*Vídeo Annotation for Speech Technologies*), audio extraído de grabaciones de vídeo de internet con los siguientes retos: compresión de audio, diversos temas, sistemas de grabación y ambientes.

También se han usado datos de la colección IARPA Babel, la cual contiene grabaciones de pobres recursos. Se caracteriza por contener distintos lenguajes y de conversaciones telefónicas. Estos últimos (*Conversational Telephone Speech*, CTS) son los usados en este proyecto.

Por último, como evaluación se ha hecho uso de un conjunto de datos de Public Safety Communications (PSC), en particular, de audios de el incendio de Sofa Super Store (*Sofa SuperStore Firefighters*, SSSF) que ocurrió en junio de 2007 en Charleston, Carolina del Sur. Estos audios presentan, en general, ruido de fondo, efectos de transmisión de radio y habla bajo estrés físico y cognitivo.

4 Diseño y Desarrollo

4.1 Introducción

Una vez expuesto en el *Estado del Arte* el estudio que se ha hecho previo al desarrollo del proyecto para la adquisición de conocimientos acerca del tema se va a proceder a explicar qué y por qué se han elegido la estructura de cada red y las características de entrada, además de qué tipo de evaluación se ha hecho.

El desarrollo de este Trabajo Fin de Grado se puede dividir en dos partes diferenciadas por el tipo de red neuronal utilizada para la experimentación. Dentro de cada de una de estas partes se pueden diferenciar tres etapas: la preparación previa al entrenamiento, el diseño de la red neuronal para el entrenamiento y el propio entrenamiento que se ha decidido llevar a cabo. Además, se incluye en este apartado una descripción de la técnica de evaluación utilizada.

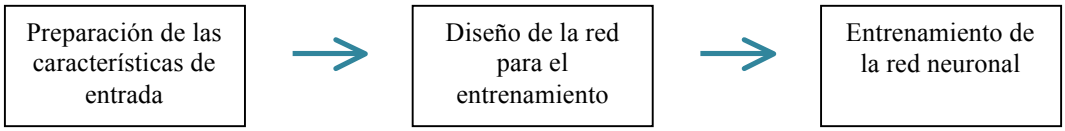


Figura 4-1: Esquema del Diseño y Desarrollo para cada red

4.2 Red DNN

La red principal en la que se basa el proyecto es la Red Neuronal Profunda, la cual comenzó a utilizarse para procesos de voz en el ámbito del reconocimiento **[31].** Como ya se ha explicado anteriormente se trata de una red de propagación "hacia delante" (*feed-forward*), es decir, las conexiones van desde la capa de entrada a la capa de salida. Esta elección de modelo se basa en que, principalmente, es uno de los tipos de redes neuronales más sencillas de entender y de implementar en cuanto a entrada de características y arquitectura en sí. Además, en otros trabajos de investigación **[32]** se han obtenido mejores resultados de precisión que utilizando que utilizando modelos clásicos como GMM.

4.2.1 Preparación previa

Para caracterizar la información de los audios de las bases de datos, se han utilizado los MFCCs descritos anteriormente debido a su extendido uso en los casos de estudio existentes y, en general, en el procesamiento de voz. Para este proyecto estas características se han dado ya calculadas y, a continuación, se hablará de su acondicionamiento previo a la entrada de la red neuronal.

Se tienen ficheros de extensión ".mat" con estas características y las etiquetas asociadas. En cuanto a las primeras, se tiene una matriz de 46654x60, es decir, 60 MFCCs para cada muestra del segmento a entrenar. Y de etiquetas se tiene un vector 46654x1, una etiqueta (1, voz y 0, no voz) por muestra del segmento. El número de ventanas (*frames*) por segmento variará dependiendo de su duración; sin embargo, para todos se han extraído 20 MFCCs (aumentados con sus derivadas de primero y segundo orden), aunque en este trabajo sólo se utilizan los 20 MFCCs propiamente dichos.

Los datos de las figuras anteriores corresponden al audio de voz "*BABEL_BP_104_01820_20120313_212614_inLine.wav*" perteneciente a la base de datos de OpenSAT, ya descrita. La forma de onda de este archivo se representa como muestra la **Figura 4-2,** en la cual se observa que existen zonas de silencio entre las zonas de sonido.

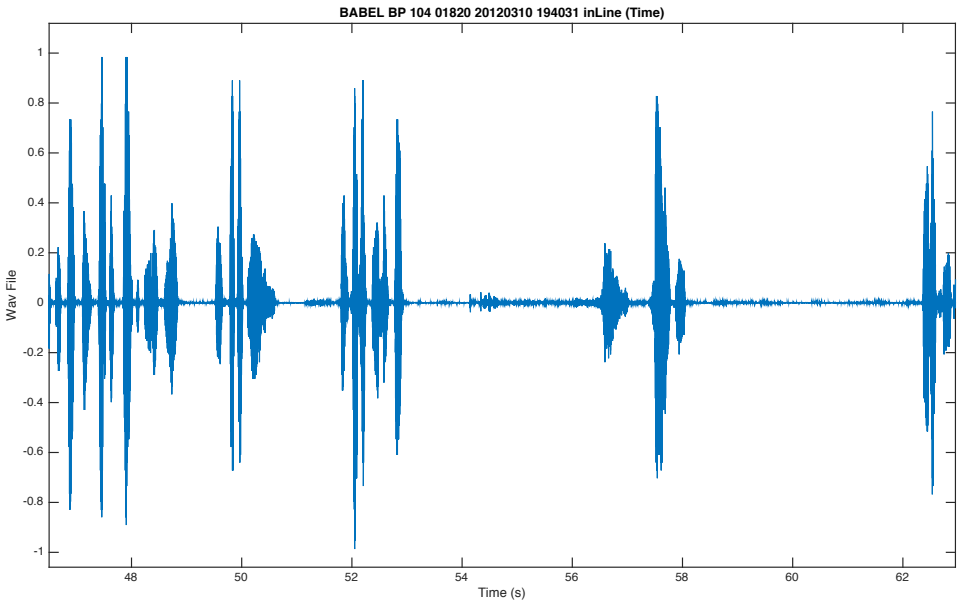


Figura 4-2: Ejemplo de unos segundos de la forma de onda de un archivo. De él se extraen características y etiquetas

Se han utilizado 1278 archivos, cada uno correspondiente con un audio de la base de datos. Estos 1278 archivos se han dividido en varios conjuntos para ir utilizándolos por partes durante el entrenamiento de la red por limitaciones de memoria.

De cada archivo se han tomado 50 segundos (5000 *frames*) de muestra con comienzo aleatorio. Para su entrada a la red, cada vector de 20 MFCC se ha pre-procesado con un contexto de 15 tramas (*frames*) a cada extremo (resultando en vectores de características de dimensión 20 MFCC x 31 tramas = 620). Este contexto es característico de las DNN ya que, como son redes *feed-forward,* solamente pasan por los datos una vez sin volver a ellos, y por tanto necesitan de cierta información anterior y posterior para optimizar su rendimiento. Aunque podría existir un beneficio mayor al incorporar ventanas de contexto mayores, se han probado distintos valores entre los 21-81 *frames* propuestos en **[19]** y se ha obtenido un mayor error comparado con los 15 definidos inicialmente. En este procesamiento de las características se obtienen matrices de entrada a la red de 4970x620 por fichero.

Para las etiquetas se ha realizado el mismo proceso, tomando las correspondientes a los 50 segundos de comienzo aleatorio sin incluir información adicional en forma de 15 *frames* de contexto anterior y posterior, ya que se tiene de datos anteriores y posteriores (es decir, se utiliza la etiqueta del *frame* central). Por tanto, se consigue un vector de 4970x1, aproximadamente.

Finalmente, dichas características ya procesadas se almacenan en ficheros de formato Python para su lectura posterior en el entrenamiento de la red.

Para las posteriores pruebas en datos distintos, se obtienen media y desviación típica de las características de *train* para normalizarlas. Esto es necesario para que las salidas de los conjuntos de datos se encuentren en rangos comparables. Además, las etiquetas se ponen en formato *one-hot vector* de forma que para la clase 0 se obtendría un vector [1 0] y para la clase 1, [0 1], es decir, un 1 en el índice del vector para representar a cada clase (necesario para la implementación en Keras de la función de coste utilizada). De esta forma, se obtendrá como salida cierto porcentaje de pertenencia a cada clase sumando 1 aproximadamente.

4.2.2 Diseño de la Red Neuronal

Se han hecho diversas pruebas, de las cuales se describirán las más relevantes para el apartado *Pruebas y Resultados* para finalmente obtener la red entrenada óptima en cuanto a resultados.

Como ya se ha definido antes, la DNN es una red de varias capas ("*deep*") *fullly connected* o, en castellano, neuronas completamente conectadas unas con otras. Para el caso de estudio se ha variado el número de capas de 4 a 5, esto es, una capa de entrada, 2 ó 3 ocultas y una de salida. La mayoría de pruebas se han realizado sobre la red de 3 capas ocultas debido a sus mejores resultados, pero se quiere recalcar la diferencia en el siguiente apartado.

Por otro lado, la capa de entrada tiene una dimensión de 620, correspondiente a la dimensión de los vectores de características definidos anteriormente. Las siguientes tres capas ocultas presentan el mismo tamaño elegido de 500 neuronas.

Típicamente, los estudios que se encuentran en la literatura sobre el número de noches que debería tener cada capa oculta, recomiendan que sea un balance entre el tamaño de la capa de entrada y de salida, así como el número de datos de entrenamiento disponible. De esta forma se conseguiría minimizar el error y tener una red entrenada que generalice correctamente. Muy pocos nodos o neuronas pueden llevar a un error alto ya que los factores de predicción pueden ser demasiado complejos para ellos, y el modelo demasiado simple. Sin embargo, un número demasiado grande de nodos generará un modelo demasiado complejo y producirá, probablemente, *overfitting* y no generalizará como debe. En **[1]** se detalla cómo se encuentra un número óptimo de neuronas para cada capa que, aunque sea específico para su problema, es interesante para entender la elección.

Por último, la capa de salida cuenta con dos neuronas, una por clase.

El esquema de la estructura del grafo final se muestra en la **Figura 4-3**.

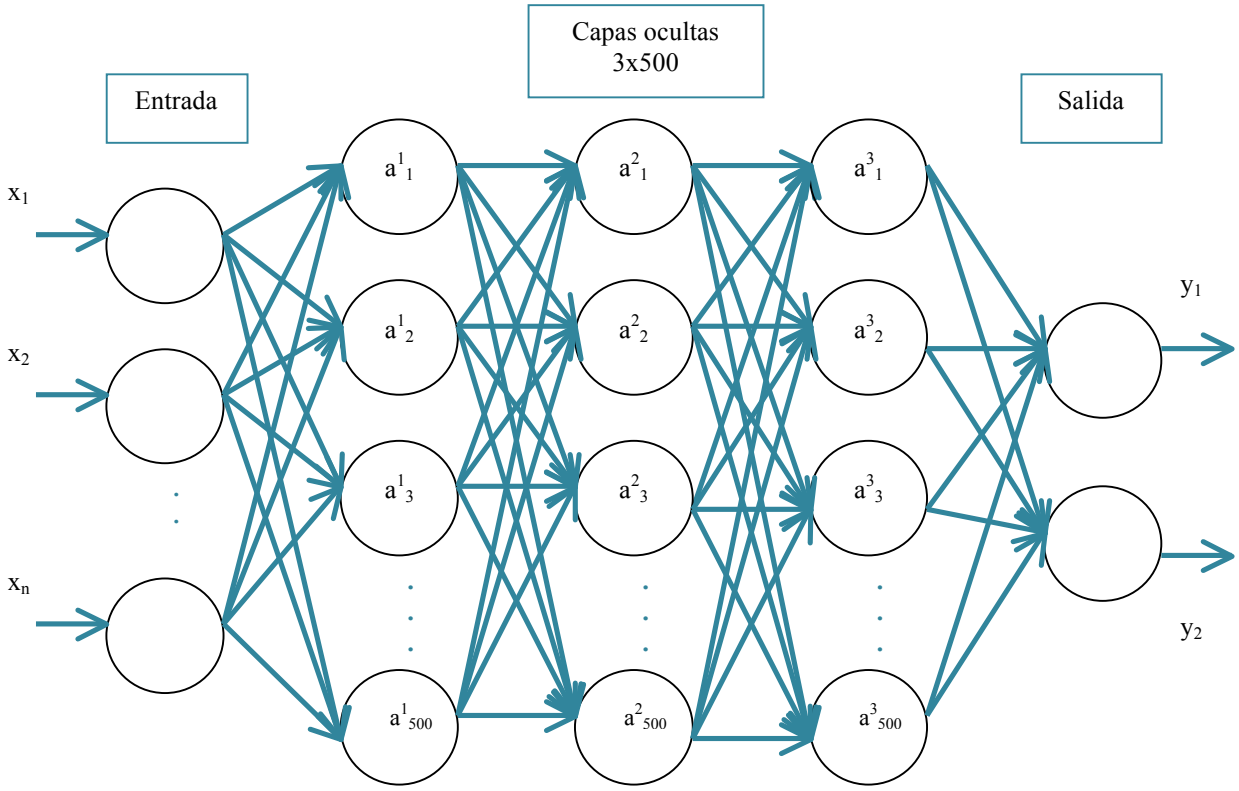


Figura 4-3: Estructura DNN

Los pesos que se calculan para los datos en cada unión entre capas (en este caso 4) son matrices de tantas filas como número de neuronas que tiene la capa de donde vienen y columnas como la capa a la que se dirigen. En la primera capa se tendría una matriz de pesos de 620x500, según lo explicado.

Para finalizar el apartado, se hará una breve explicación acerca de las funciones de activación elegidas. Como ya se ha dicho, esta función permite la combinación de transformaciones no lineales de los datos de entrada. En este caso, se ha decidido utilizar la *sigmoid*. Además, la restricción de valores al rango [0, 1], como hace la función *sigmoid*, hace que los valores de entrada a la siguiente capa estén acotados.

Como ya se ha dicho el valor de la salida de la neurona calculada en base a pesos y entradas puede variar entre -∞ y +∞, la neurona no conoce los límites de este valor, por tanto, se le aplica una función de activación para permitir la combinación de transformaciones. En este caso de estudio, necesitamos valores en el rango [0,1], por ello, la función elegida es la sigmoidea, la cual elige resultados comprendidos entre 0 y 1. Esta función se usa para las tres capas ocultas, mientras que para la capa de salida se usa la *softmax*. La función *softmax* se suele usar en la última de capa de las redes neuronales que aprenden un clasificador. Esto se debe a que normalmente, y en este caso, estas redes se suelen entrenar bajo un régimen de cálculo de entropía cruzada (*cross-entropy*), dando una variante no lineal de una regresión logística multi-nominal (multi-clase). Por tanto, esta función se utiliza para representar esta distribución categórica obteniendo probabilidades de clase para cada salida (que suman 1).

4.2.3 Desarrollo del Entrenamiento

Para el entrenamiento de esta red se ha hecho uso de la GPU, que presenta un mayor rendimiento en cuanto a velocidad de procesamiento con respecto al cómputo en CPU. Se trata de una tarea supervisada para la clasificación en voz/no voz en la que se obtendrán unos pesos (parámetros), como se ha dicho anteriormente, y se busca obtener el mejor valor de precisión en la clasificación final.

Dado que se tienen por archivo bastantes muestras de entrenamiento, se divide la entrada en lotes, es decir, en *batches*. Esto requiere mucha menos memoria al entrenar, por tanto, irá mucho más rápido. Además, de esta forma se actualizan los valores de los pesos al acabar el entrenamiento de cada *batch*; de otra forma, solamente se actualizarían una vez En este caso, se ha utilizado un tamaño de *batch* de 512.

En cuanto al número de épocas, es decir, número de veces que se ha pasado por todos los datos de entrenamiento durante el entrenamiento, se ha decidido usar unas 20 iteraciones, pero, como ya se verá más tarde es posible que no se necesiten tantas para ver resultados óptimos. Cuantas más *epochs* se elijan, más tiempo se estará ejecutando el entrenamiento, lo cual no siempre es bueno ya que la red podría llegar al sobre-entrenamiento.

Para la compilación de la red no se ha fijado un optimizador para minimizar o maximizar la función de coste, si no que se ha probado entre dos posibles y, más adelante, se desarrollará el porqué del uso de cada uno. En cuanto a las métricas para monitorizar, se ha decidido observar la *accuracy* o precisión al ser un problema de clasificación, para comparar cada iteración y cada red. Por último, se encuentra la función de coste. Al ser una clasificación binaria, dos clases, se ha decidido utilizar la entropía cruzada binaria. En el *Estado del Arte* se explicó en qué consistía la *cross-entropy;* la binaria simplemente se utiliza cuando son dos categorías o clases.

4.3 Red LSTM

Además del uso de una red DNN como la explicada anteriormente, dicha arquitectura se ha querido comparar con otro tipo de red neuronal. En este caso se ha elegido una red recurrente (*Recurrent Neural Network*, RNN), ya que las DNNs solo pueden depender del tiempo limitado por la ventada fijada junto con el contexto en el vector de entrada. Como se explicaba anteriormente en el *Estado del Arte,* al contrario que la arquitectura *feed-forward* de las redes profundas, las recurrentes pueden aprender complejas secuencias dinámicas de tiempo. Específicamente, se ha elegido una LSTM debido a su habilidad para modelar grandes rangos de dependencia entre las entradas. En la **Figura 4-4,** se muestra una comparación de los esquemas de las estructuras de los dos grafos, DNN y LSTM.

Red Neuronal Recurrente **Red Neuronal Profunda (*Feed-Forward*)**

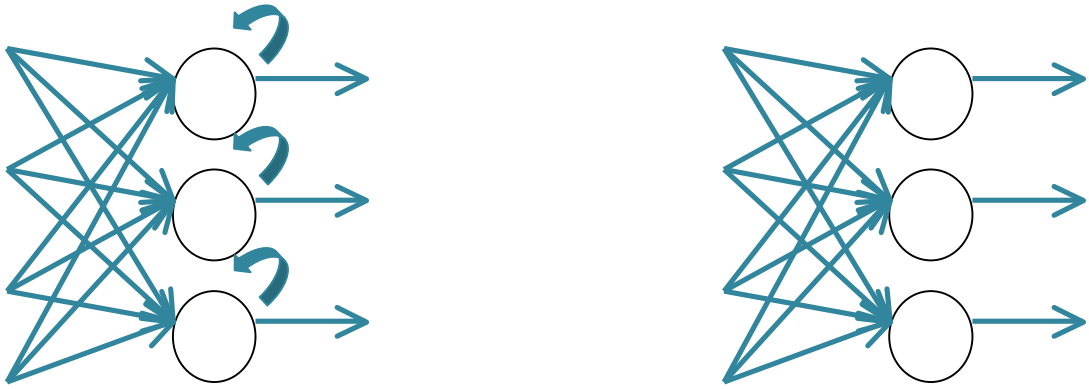


Figura 4-4: Comparación estructuras LSTM y DNN

Debido a que se ha incluido una explicación acerca de la otra red sobre la que trata el proyecto en el punto anterior, en los siguientes apartados tan solo se harán incisos en las diferencias en implementación entre ambas. Se intentará mostrar una diferencia gráfica entre ambas respecto a distintos aspectos hablados.

4.3.1 Preparación previa

Se han utilizado las mismas características de entrada descritas en **4.2.1** al igual que las mismas etiquetas. Sin embargo, esta red tiene como característica principal que presenta como entrada un conjunto de secuencias de datos, se trata de una entrada tridimensional.

Se han tomado 50 segundos (5000 *frames*) como en la DNN, pero estos segundos se han dividido en secuencias. De tal forma, en lugar de tener una matriz de dos dimensiones de 5000x20 (*frames* x MFCCs), se ha decidido dividir esa primera dimensión en secuencias de 300 *frames* o 3 segundos. Si, por ejemplo, la duración total es de 9 segundos, se crearían 3 trozos de 3 segundos cada uno, descartando el último trozo si hay más de uno o rellenándolo con ceros si el fichero es más corto de 3 segundos.

De esta forma se tendría una dimensión final de características de entrada para cada archivo de la lista de audios de 16x300x20, ya que 300x16=5000 segundos.

En cuanto a las etiquetas de los archivos, ya que para cada *frame* la etiqueta puede ser 0 ó 1, es decir, no existe el mismo valor de etiqueta para cada archivo, se van a dimensionar las etiquetas de la siguiente forma y se trabajará con salidas a nivel de *frame*. Se toman las etiquetas respectivas a los 50 segundos de características tomadas y se hace una transformación acorde a las secuencias generadas, de tal forma que se obtendrían 16x300, es decir, por cada secuencia habría 300 etiquetas haciendo un total de 1 etiqueta por cada *frame*: 5000 etiquetas por archivo. En la **Figura 4-5** se muestra una comparación entre las entradas a las redes profundas y a las recurrentes.

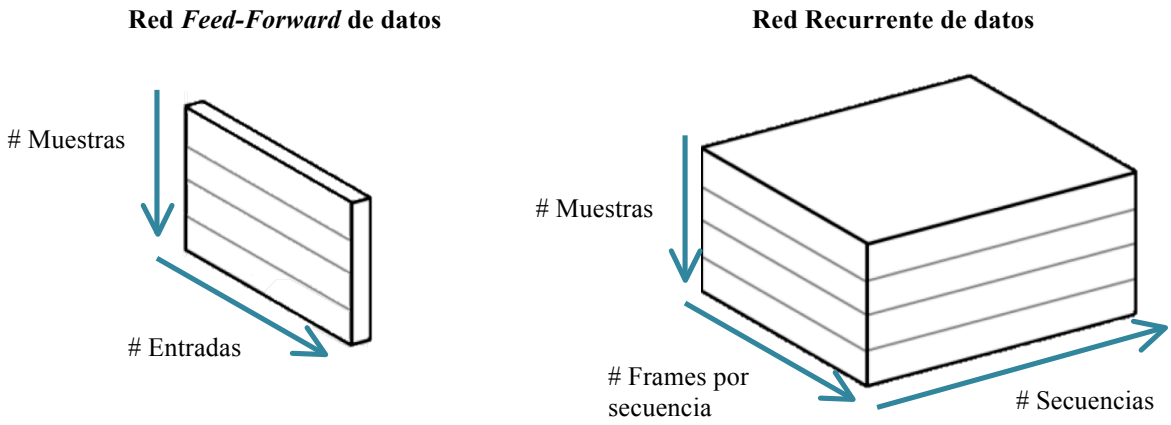


Figura 4-5: Comparación entre los datos de entrada de ambas redes

4.3.2 Diseño de la Red Neuronal

La arquitectura elegida para la LSTM se ha basado en la de la anterior red implementada. Se han ido diseñando distintas estructuras para proceder a hacer la evaluación final sobre la de mejor resultado en cuanto a precisión.

Se ha partido de una implementación de 5 capas: la de entrada, la de salida y tres ocultas. Las tres ocultas son 3 LSTM con 500 neuronas cada una, recibiendo la primera una dimensión de tamaño de *batch* por número de secuencias por dimensión de las características, MFCCs (512x300x20). La capa de salida presenta una neurona por clase (2 clases) y se trata de una DNN, pero con un *wrapper* de tipo *TimeDistributed*. Este *wrapper* aplica la función de la capa a toda unidad temporal de su entrada.

Además, es importante recalcar que se ha implementado la red para que, en lugar de que se proporcione la salida del último paso de entrenamiento, se devuelva toda la secuencia de salidas calculadas en la *cell*. En la **Figura 4-6** se muestra gráficamente lo que supondría.

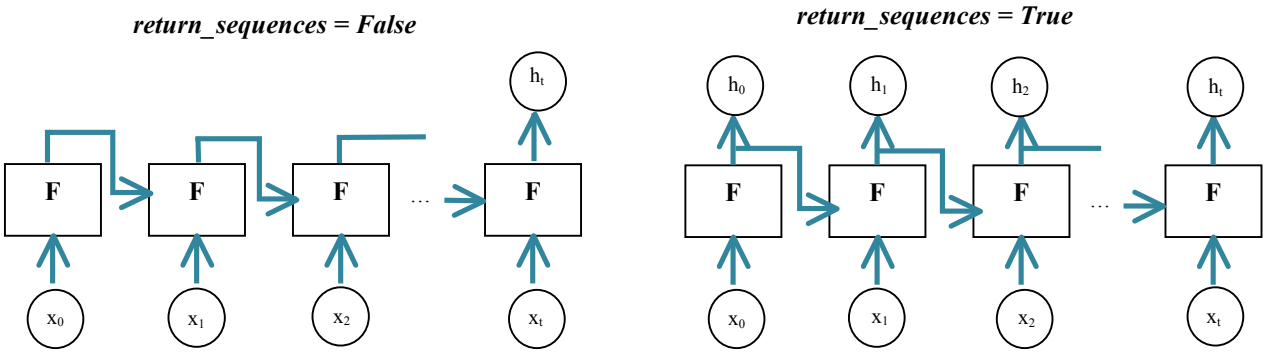


Figura 4-6: Comparación entre salidas de la red LSTM

4.3.3 Desarrollo del Entrenamiento

Tanto las funciones de activación de las primeras capas y de la última como la de coste no han variado en la implementación de esta red con respecto a la DNN. Sí que hay que recalcar el uso del optimizador en la configuración del modelo. En este caso, el optimizador apropiado para las LSTMs suele ser RMSprop, cuyos parámetros por defecto se han mantenido salvo algunas pruebas con variaciones de la tasa de aprendizaje.

En cuanto al proceso de aprendizaje o entrenamiento no hay mucho que difiera del de la red DNN, ya que el número de *batches* e iteraciones ha sido el mismo. Simplemente recordar que esta red contiene información presente y pasada, es decir, tiene memoria y puede leerla, escribirla y borrarla. Por tanto, cada *cell* decide si guarda o elimina información basándose en la importancia que tenga esta, la cual proviene de los pesos aprendidos durante el algoritmo.

4.4 Medidas de Evaluación

En este último apartado se definirá cuál será el proceso de evaluación de resultados obtenidos para la posterior comparación. Se va a seguir, por una parte, la evaluación de la tasa de error igual (*Equal Error Rate*, EER). Por otra parte, se quiere hacer una comparación entre matrices de confusión y *accuracy,* así como una demostración visual de la salida frente a la forma de onda del audio.

4.4.1 *Equal Error Rate* (EER)

Lo que se busca para la salida del sistema es evaluarla en base a una comparación con las etiquetas verdaderas de voz o no voz que se tienen de antemano. Las probabilidades de error del sistema se determinarán, entonces, en base a segmentos correctos, incorrectos o parcialmente correctos.

En la **Figura 4-7** se muestra la relación entre los objetivos reales, una posible salida del sistema con los scores obtenidos y, por último, los intervalos de tiempo clasificados como verdaderos negativos (*true negative*, TN), verdaderos positivos (*true positive*, TP), los falsos negativos (*false negative*, FN) o *miss* y los falsos positivos (*false positive*, FP). Los FN determinan una detección de no voz cuando en realidad la hay y los FP, detección de voz cuando esta no existe.

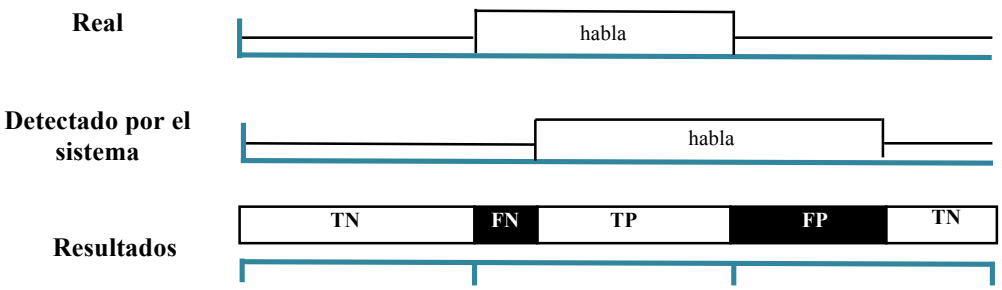


Figura 4-7: Comparación target y predicción

En este caso, en el cual no se quiere tener que elegir un umbral, se medirá la tasa de error igual (*Equal Error Rate,* EER). El EER es un valor que indica la proporción de falsos positivos coincidentes con falsos negativos. Cuanto más pequeño sea este valor, mayor será la precisión de este sistema.

Se tienen dos funciones que calculan, por una parte, los falsos negativos, tratados a partir de ahora como P_{Miss}, y los falsos positivos, P_{FA}, de la detección en base a los valores reales y, por la otra, a partir de estos valores obtenidos, el EER.

Básicamente, se calcularán P_{Miss} y P_{FA} de la siguiente forma representada en la **Figura 4-8**, teniendo en cuenta que FN son falsos negativos y FP, falsos positivos. Por otra parte, *tiempo total voz* tendrá que igualar a la suma de *tiempo FP total* y *tiempo FN total*. Y, por último, *tiempo total predicho como no voz* consiste en la suma de *tiempo FP total* y TN.

$$\text{Falso rechazo } (P_{Miss}) = \frac{tiempo\ FN\ total}{Tiempo\ total\ voz}$$

$$\text{Falsa aceptación } (P_{FA}) = \frac{tiempo\ FP\ total}{Tiempo\ total\ predicho\ como\ no\ voz}$$

Figura 4-8: Ecuaciones de P_{miss} y P_{FA}

Los métodos para calcular P_{Miss} y P_{FA} y EER han sido proporcionados, haciendo únicamente el cálculo de los "true scores" y los "false scores". Ambos se han calculado en base a las probabilidades obtenidas para la salida de voz y las clases reales.

Por tanto, los errores de VAD se miden como la cantidad de veces que se han clasificado erróneamente los datos de cada segmento de los archivos de audio.

4.4.2 Matriz de Confusión y *Accuracy*

En el ámbito del aprendizaje automático supervisado, para visualizar la tarea de un algoritmo se utiliza una matriz de confusión. La matriz de confusión muestra las formas en las que el modelo de clasificación creado está equivocado al hacer predicciones. Por tanto, es un buen método para comparar ambas redes ya que se comparan ambos comportamientos. En esta matriz cada fila representa el número de predicciones de cada clase y cada columna representa las instancias en la clase real.

Esta matriz almacena todas las clasificaciones hechas explicadas anteriormente: verdaderos positivos (TP), verdaderos negativos (FP), falsos positivos (FP) y falsos negativos (FN), tal y como se muestra en la **Figura 4-9.** Si se tiene la lista de predicciones y la de valores esperados se puede calcular el número de predicciones correctas para cada clase, así como el número de predicciones incorrectas para cada clase organizadas por valor predicho. Con estos valores se podría crear una matriz de confusión.

		Clase Correcta	
		Positivo	**Negativo**
Clase Predicha	**Positivo**	Positivo Verdadero	Positivo Falso
	Negativo	Negativo falso	Negativo Verdadero

Figura 4-9: Relaciones en una Matriz de Confusión

Por último, uno de los valores recogidos del evaluador del sistema de la librería Keras al pasarle los datos de prueba ha sido el *accuracy* o precisión del sistema de predicción. Para clasificación binaria, Keras calcula este porcentaje de exactitud como lo representado en la **Figura 4-10.** Esta fórmula sugiere que el umbral es 0,5 para diferenciar entre las clases, es decir, algo mayor a 0,5 será considerado como perteneciente a esa clase.

```
def  binary_accuracy (y_true, y_pred):
        return K.mean (K.equal (y_true, K.round (y_predict)), axis = -1)
```

Figura 4-10: *Accuracy* binaria calculada en Keras

Para este valor de precisión, se ha calculado en un principio si las clases voz y no voz estaban balanceadas en los datos de entrenamiento y de pruebas. Esto se ha hecho porque en problemas de clasificaciones es muy posible que las distintas clases no estén igualmente representadas. En el caso de que se tuviese en una clasificación binaria 100 instancias en las que 80 fuesen de la clase 1 y el resto de la clase 0, se tendría una balanza 4:1 y el sistema podría obtener un 80% de *accuracy* simplemente decidiendo clasificar todas las muestras como clase 1. En este caso, se ha comparado el porcentaje de voz y no voz en los datos de entrenamiento obteniendo un 49,28% de voz y un 50,72% de no voz, por tanto, no se consideran desbalanceados los datos. En el caso de los datos de validación y de test, se han obtenido unos porcentajes de voz de 40,61% y 41,87%, respectivamente, así que se puede tener en cuenta el valor de *accuracy* obtenido. En la **Figura 4-11** se muestra una distribución de las clases para cada conjunto de datos utilizados.

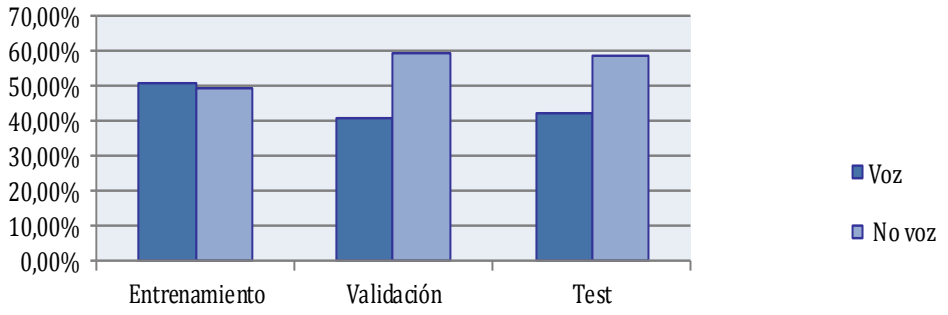


Figura 4-11: Proporción de los conjuntos de datos utilizados

5 Pruebas y resultados

En los siguientes apartados se van a describir los experimentos más relevantes realizados en este trabajo, junto con los resultados. Cada una de estas pruebas busca mostrar los distintos comportamientos de las redes en función del diseño de la arquitectura (hiper-parámetros), que se resumen en la **Tabla 5-1.**

Exp	Red	Capas Ocultas	Dropout	Optimizador
1	DNN	2	Sí	SGD
2	DNN	3	Sí	SGD
3	DNN	3	No	SGD
4	DNN	3	No	Adam
5	LSTM	3	No	RMSprop
6	LSTM	3	No	Adam
7	LSTM (variación)	3	No	Adam

Tabla 5-1: Experimentos realizados

Para cada *Experimento* resumido en la tabla anterior, se han obtenido los siguientes resultados mostrados en la **Tabla 5-2.** Para su cálculo se han tomado los pesos de la iteración en la que mejores resultados de *accuracy* para los datos de validación se hayan obtenido. Estos se irán desglosando en los siguientes apartados.

	EER (%)	***Accuracy* (%)**
Experimento 1	16,9081	82,30
Experimento 2	26,4167	73,72
Experimento 3	13,7144	81,42
Experimento 4	19,2848	76,73
Experimento 5	27,5667	78,33
Experimento 6	31,9838	75,79
Experimento 7	21,9227	85,25

Tabla 5-2: Resultados obtenidos

Para la realización de las pruebas de validación y test se han utilizado datos de la base de OpenSAT. Al igual que para los datos de entrenamiento, se tenían archivos de extensión ".mat" con las características y las etiquetas extraídos previamente. La preparación anterior a la evaluación de estas características y etiquetas ha sido similar a la de los archivos de entrenamiento. Sin embargo, en lugar de tomar 50 segundos aleatorios, se han tomado

todas las muestras de características extraídas de cada archivo dimensionándolas para la entrada de cada red.

El conjunto de datos de validación es el mostrado en **Tabla 5-3,** dado que para entrenamiento se tenían unos 7.000.000 de muestras, para validación se han cogido simplemente los mostrados ya que un número bastante menor permite una validación más rápida. De las medidas obtenidas para cada conjunto en cada época, se seleccionan las que representan un mejor rendimiento del sistema.

Set de Datos	Cantidad de Muestras
1	917.870

Tabla 5-3: Cantidad de muestras de validación

Por otro lado, para evaluar las arquitecturas planteadas, tanto DNN como LSTM, se va a utilizar el mismo conjunto de datos de test. De la base de datos proporcionada, se ha decidido utilizar un conjunto de datos que proporciona imparcialidad a la hora de evaluar el rendimiento del sistema, puesto que no ha sido utilizado para su entrenamiento. Este conjunto presenta la cantidad de muestras mostrada en la **Tabla 5-4.**

Set de Datos	Cantidad de Muestras
1	180.014

Tabla 5-4: Cantidad de muestras de test

5.1 Experimentos en DNN

Para comenzar las pruebas con la red DNN se ha partido de un modelo de base simple como el descrito en *4.2.2 Diseño de la Red Neuronal.*

El resumen del modelo es el mostrado en la **Tabla 5-5** habiendo una variación en número de capas para el primer experimento.

Capa (tipo)	Dimensiones de salida	Parámetros
Dense_1 (Dense)	(None, 500)	310.500
Dense_2 (Dense)	(None, 500)	250.500
Dense_3 (Dense)	(None, 500)	250.500
Dense_4 (Dense)	(None, 2)	1.002

Tabla 5-5: Resumen modelo DNN

Se ha comenzado con un modelo simple compuesto por 2 capas ocultas. Para estos experimentos destacables de entre los realizados se ha entrenado durante 20 épocas para ir viendo cómo mejoraba el rendimiento. La función de optimización utilizada, en un principio, ha sido la SGD (*Stochastic Gradient Descent*, uno de los algoritmos más

populares para optimización en redes neuronales) con una tasa de aprendizaje de 0.008 y, además, se han implementado las capas con *dropout* del 20% para evitar un posible *overfitting*. Este último valor quiere decir que una de cada 5 entradas se excluye del ciclo de actualizaciones.

Experimento_1: Para empezar, se ha entrenado la red descrita. A la hora de elegir el número de capas, un número menor conlleva un modelo con menos pesos y, por tanto, un modelo menos complejo de red. En la **Figura 5-1** se puede comprobar que los datos de pruebas presentan casi un 80% de *accuracy* con este modelo simple. Es posible que el problema, junto a los datos, sea sencillo de resolver y, por tanto, un modelo sencillo como este funcione correctamente.

Las representaciones de curvas de *accuracy* para los experimentos siguientes seguirán la misma relación en cuanto a validación y entrenamiento, por ello no se entrará a fondo en cada una a menos que haya algo representativo.

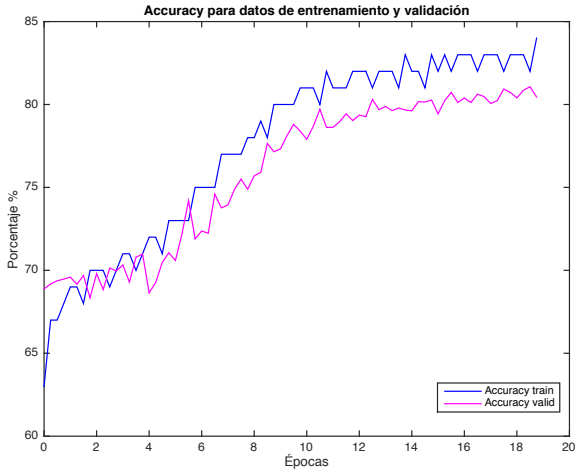


Figura 5-1: Curvas de *accuracy* para el Experimento 1

En la evaluación con el conjunto de datos de test se ha obtenido un 82,30% de *accuracy* y un EER del 16,9081%. En cuanto al *accuracy* obtenido, es menor que para el mejor resultado del conjunto de validación, ya que los datos presentan características distintas por lo que le es más complicado a la red clasificarlos. Ocurre lo mismo para los siguientes experimentos.

La **Tabla 5-6** muestra la matriz de confusión para este experimento. Para los datos de test, un 13,02% de etiquetas de no voz han sido clasificadas como voz y un 6,95% de etiquetas de voz han sido clasificadas como no voz. En los problemas de detección de voz es preferible tener un número pequeño de muestras de voz mal clasificadas. Esto se debe a que se produce una pérdida de información importante. Uno de los motivos de mal etiquetado en muestras de no voz suele ser a que pueden ser zonas del audio en el que se va a cambiar de silencio a voz o viceversa y, por ello, se confunde el sistema.

Para los siguientes experimentos se sigue el mismo patrón en la matriz de confusión.

		Clases reales	
		0	**1**
Clases predichas	**0**	42,51%	13,02%
	1	6,95%	37,52%

Tabla 5-6: Matriz de Confusión para el Experimento 1

Por último, para este experimento se va a mostrar en la **Figura 5-2** la relación entre 50 segundos de la forma de onda de un archivo del conjunto de evaluación y las probabilidades para cada clase de la salida del sistema. Además, se muestran las etiquetas reales para comparar junto con las obtenidas a partir de las probabilidades. Para el resto de experimentos las representaciones se encuentran en el *Anexo*.

En la tercera gráfica de la representación se observan las probabilidades de clases obtenidas a la salida. Son probabilidades dispares, propio de la DNN, ya que luego se observará en una LSTM que estas probabilidades son más "suaves". Debido a que, como se ha dicho anteriormente, se trata de un audio de una llamada de emergencia, es decir, en un ambiente complicado de estrés, la última gráfica muestra cómo se confunden zonas de no voz por voz (son datos a clasificar complicados).

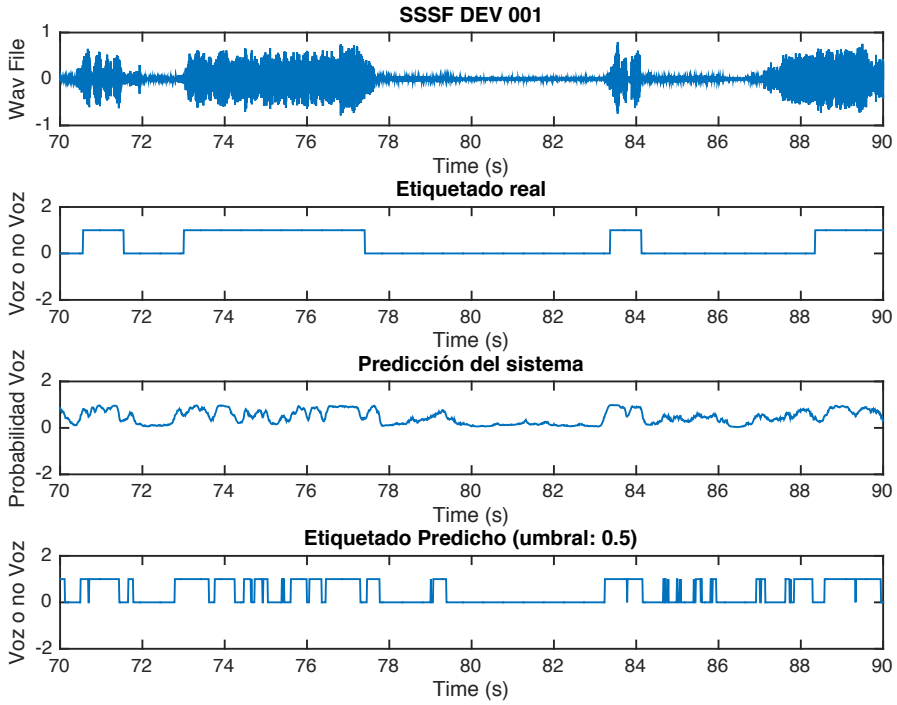


Figura 5-2: Forma de onda y etiquetado real comparados con predicciones del Experimento 1

Experimento 2: Dado que para el primer experimento se ha entrenado con una red simple, se ha decidido añadirle una capa para hacer el modelo más complejo. En la **Figura 5-3** se observa un comportamiento similar al anterior en cuanto a la relación entre conjunto de entrenamiento y de validación. Sin embargo, en este caso se ve que al añadir una capa más,

tarda más en aprender. Por lo tanto, para alcanzar un mismo valor de *accuracy* se necesitaría entrenar durante más épocas. Esto se debe principalmente a que la red tarda más en adaptarse por la presencia de un mayor número de parámetros.

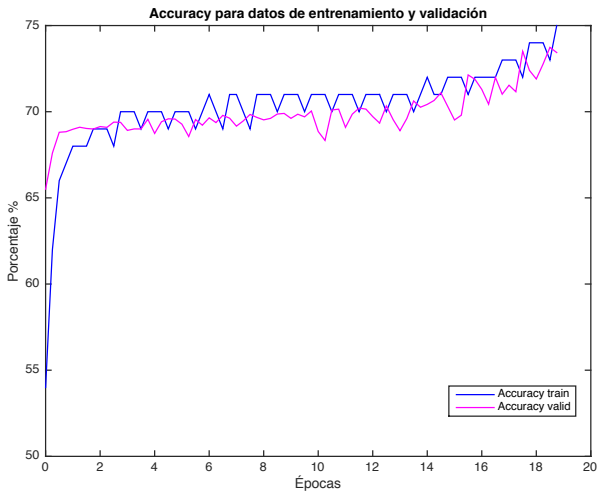


Figura 5-3: Curvas de *accuracy* para el Experimento 2

En este experimento, para los datos de evaluación se ha obtenido un 73,72% de *accuracy* y un EER de 26,4167%. En la **Tabla 5-7** se ve un pequeño empeoramiento en las clasificaciones, pero probablemente si se entrenase durante más tiempo, habría mejores resultados.

		Clases reales	
		0	**1**
Clases predichas	**0**	38,17%	17,37%
	1	8,84%	35,62%

Tabla 5-7: Matriz de Confusión para el Experimento 2

Experimento 3: Dado que se comenzó usando un *dropout* del 20% sin saber si sin él habría *overfitting*, se ha querido probar el modelo sin esta regularización. En muchos casos, el *dropout* beneficia al rendimiento, pero, en otros, puede hacer más complicado su entrenamiento. Es posible que, para el conjunto de datos de entrenamiento, la red sea relativamente pequeña y, por ello, la regularización no haga falta. También puede ser que, si se hubiese esperado a que el error convergiera, este hubiese resultado menor dado que el *dropout* afecta al principio del entrenamiento normalmente. En la **Figura 5-4** se puede observar una mejora para ambos valores, por tanto, se decide dejar de usar a partir de este experimento y evaluar la influencia de otros hiper-parámetros. A la vista de los resultados se puede confirmar que se estaba produciendo el caso contrario, *underfitting*.

Para los datos de test en el *Experimento 3* los datos de *accuracy* y EER obtenidos son de un 81,42% y un 13,7144%, respectivamente. En la **Tabla 5-8** se observa una mejora respecto al experimento anterior en cuanto a clasificaciones de la red.

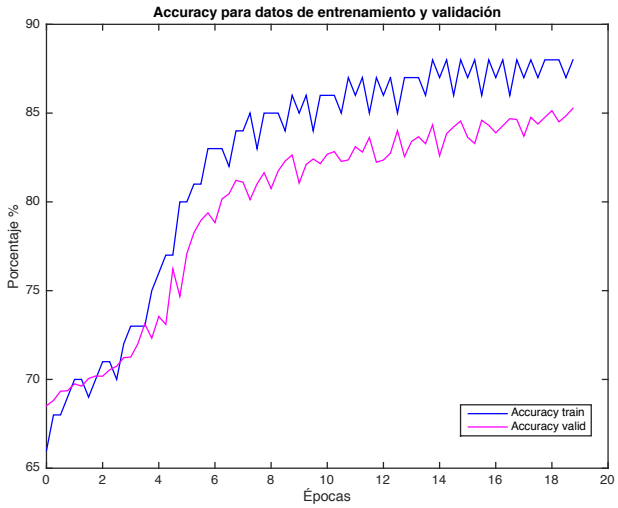


Figura 5-4: Curvas de *accuracy* para el Experimento 3

		Clases reales	
Clases predichas		**0**	**1**
	0	36,87%	18,66%
	1	2,27%	42,19%

Tabla 5-8: Matriz de Confusión para el Experimento 3

Experimento 4: En el último entrenamiento de este modelo se ha decidido modificar con respecto al anterior el optimizador pasando a Adam (*Adaptative Moment Estimation*) con sus parámetros por defecto. Este optimizador calcula tasas de aprendizajes (*learning rates*) adaptadas para cada parámetro. Es perfecto para casos de redes neuronales profundas en los que los gradientes llegan a disminuir mucho en las primeras capas, por lo que tiene sentido incrementar el valor del *learning rate*. Este valor recomendado para el optimizador Adam es 0,001; se probó con un valor de 0,008, como se estaba utilizando en el script dado de Theano pero los resultados, aunque no variasen mucho, eran algo peores al ser un optimizador robusto a cambios.

En la **Figura 5-5** sin duda se puede ver que se trata del entrenamiento con mejor rendimiento en cuanto a *accuracy*. Sin embargo, al representar la curva de *accuracy* de los datos de validación a lo largo de las épocas, se observa que esta apenas varía. Tiene pequeñas fluctuaciones, pero se mantiene un 86%. Este, claramente, no es el comportamiento que se espera.

Por ello, tras ver este comportamiento se ha decidido cambiar ciertos parámetros de configuración dentro de este modelo, pero ninguno de ellos con éxito.

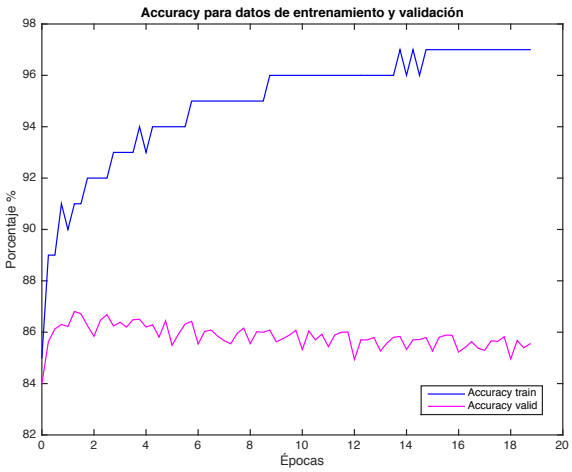


Figura 5-5: Curvas de *accuracy* para el Experimento 4

Podría tratarse de un problema de *overfitting* y que la red se hubiese adaptado a los datos de entrenamiento y para los de validación no aprendiese. Sin embargo, no se han utilizado capas de *dropout* como se ha dicho anteriormente. Otro problema podría ser el de que la tasa de aprendizaje fuese demasiado alta y no vaya a converger nunca. Por tanto, aunque este optimizador sea robusto a cambios de *learning rate*, se ha cambiado de 0,001 a 0,0001 y, aunque se hayan usado más épocas, no ha habido ningún cambio.

Dado que a partir de este momento no se ha podido mejorar la red mediante el cambio de hiper-parámetros, se ha decidido continuar con la evaluación y probar con el detector de actividad de voz con redes recurrentes, que son en general más adecuadas para el modelado de señales temporales que las DNNs.

Por último, para este experimento se ha obtenido un *accuracy* de los datos de test de 76,73% y un EER de 19,2848%. En la **Tabla 5-9** se observa un empeoramiento respecto al experimento anterior en cuanto a clasificaciones de la red.

		Clases reales	
		0	**1**
Clases predichas	**0**	33,81%	21,72%
	1	5,55%	38,91%

Tabla 5-9: Matriz de Confusión para el Experimento 4

5.2 Experimentos en LSTM

Para la red LSTM se ha hecho uso de una estructura como la que sigue. Consta de tres capas LSTM ocultas y una última capa Dense con un *wrapper* TimeDistributed, como se ha explicado anteriormente. No se ha usado *dropout* por el mismo motivo que para la red DNN. Se va a proceder a explicar los dos distintos casos en los que se han obtenido mejores resultados. Tienen que ver simplemente con el cambio de optimizador, aunque la diferencia en mejora no sea grande.

El resumen del modelo es el mostrado en la **Tabla 5-10.** Solamente basta con mirar el número de parámetros para comprobar que se trata de una red mucho más compleja.

Capa (tipo)	Dimensiones de salida	Parámetros
LSTM_1 (LSTM)	(None, 300, 500)	1.042.000
LSTM _2 (LSTM)	(None, 300, 500)	2.002.000
LSTM _3 (LSTM)	(None, 300, 500)	2.002.000
Time_distributed_1 (TimeDistributed (Dense))	(None, 300, 2)	1.002

Tabla 5-10: Resumen Modelo LSTM

5.2.1 Optimizador RMSprop

Un optimizador ampliamente utilizado para las redes neuronales recurrentes es el RMSprop, el cual utiliza la magnitud de los gradientes recientemente calculados para normalizar los actuales. Tiene varios beneficios que destacar: es un optimizador bastante robusto que presenta una pseudo-curvatura de información y, además, trabaja bastante bien con aprendizaje basado en mini-*batches* ya que se maneja bien con objetivos estocásticos.

Para este *Experimento 5* se observa en los conjuntos de entrenamiento y validación (**Figura 5-6**) que el sistema alcanza un rendimiento alto a lo largo de las épocas; sin embargo, para el conjunto de validación hay unos picos bajos en alguna de ellas. Al igual que en los experimentos anteriores, ambos conjuntos, entrenamiento y validación, presentan unos valores parecidos debido a que, aunque los de validación solo se vean al finalizar el entrenamiento, los dos conjuntos pertenecen a una misma base de datos (características similares). De hecho, hay unas zonas al comienzo de la curva donde el valor de *accuracy* del conjunto de validación supera al de los de entrenamiento, ya que la red aún no se ha ajustado a este último.

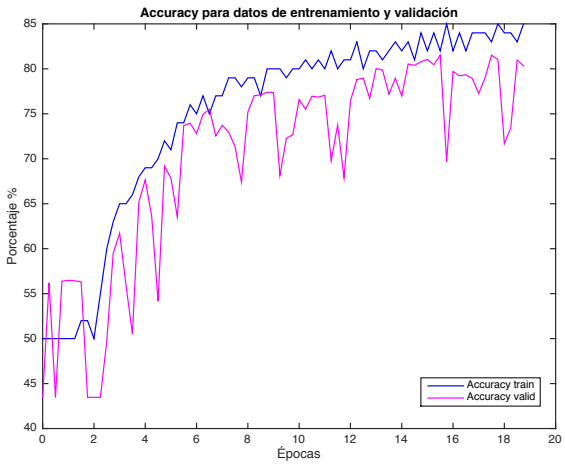


Figura 5-6: Curvas de *accuracy* para el Experimento 5

A partir de los pesos calculados en la red para la iteración con mejores resultados de validación, se ha evaluado el rendimiento con los datos de test. Por un lado, se ha obtenido un *accuracy* de 78,33% y, por otro lado, un EER de 27,5667%.

En la **Tabla 5-11** se puede observar la matriz de confusión para los datos de pruebas para esta red con optimizador RMSprop.

		Clases reales	
		0	**1**
Clases predichas	**0**	21,03%	34,50%
	1	5,97%	38,50%

Tabla 5-11: Matriz de Confusión para el Experimento 5

5.2.2 Optimizador Adam

Este optimizador utilizado para el *Experimento 6* también guarda una media exponencialmente decreciente de gradientes cuadrados pasados como RMSprop. La diferencia es que, además, guarda una media exponencialmente decreciente de gradientes pasados. Por ello, se ha comprobado que trabaja mejor que otros algoritmos de aprendizaje adaptado.

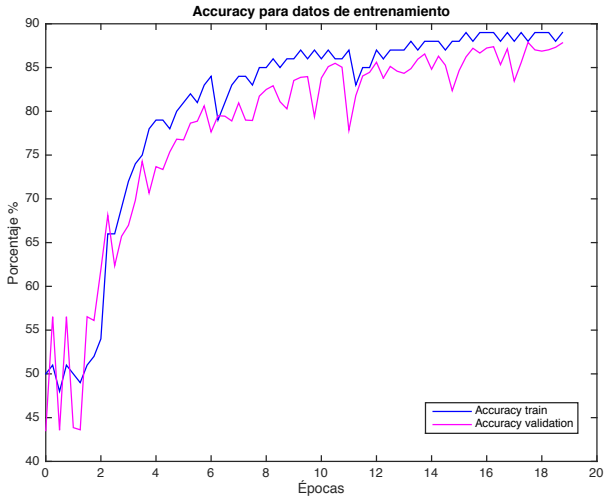


Figura 5-7: Curvas de *accuracy* para el Experimento 6

En la **Figura 5-7** se vuelve a observar que el *accuracy* de los datos de validación vuelven a superar en ciertas épocas a los de entrenamiento. En este caso de LSTM, se llega a un *accuracy* en entrenamiento y validación algo superior y, a partir de los pesos para el mejor de los casos, se obtiene el rendimiento para los datos de test. A pesar de esto último descrito, tanto el *accuracy* como la separación entre *scores target* y *scores no target* presentan valores peores que para el optimizador anterior. El *accuracy* obtenido ha sido de un 75,59% y un 31,9838% de EER.

De la matriz de confusión en la **Tabla 5-12** se observa que, para estos datos, un 31,37% de muestras de no voz han sido confundidas por voz y que un 8,84% de muestras de voz han sido clasificadas como no voz. El número de muestras de no voz mal clasificadas ha disminuido bastante con respecto al anterior caso. Sin embargo, se ha perdido más información de voz para esta clasificación.

		Clases reales	
		0	**1**
Clases predichas	**0**	24,16%	31,37%
	1	8,84%	35,62%

Tabla 5-12: Matriz de Confusión para el Experimento 6

5.2.3 Variación de la entrada de la red

Como se observa, el comportamiento del modelo utilizando como optimizador RMSprop y Adam no da los resultados esperados en comparación con la DNN. Su rendimiento debería ser mejor ya que son, en general, redes más adecuadas para el modelado de señales temporales. Es posible que el motivo principal para su peor rendimiento sea el uso de demasiados *frames* de contexto en el procesado de los datos de entrada previo al entrenamiento. Este número es de 300 *frames* y para la tarea de VAD pueden resultar demasiados, ya que se está añadiendo información innecesaria. Por tanto, para el *Experimento 7* se ha decidido generar una nueva entrada a la red con 150 *frames* de contexto. El procedimiento es el mismo pero, en lugar de 300, se utilizan 150. El diseño de la red es el mismo que para el de los apartados de LSTM y se ha decidido continuar usando el optimizador Adam.

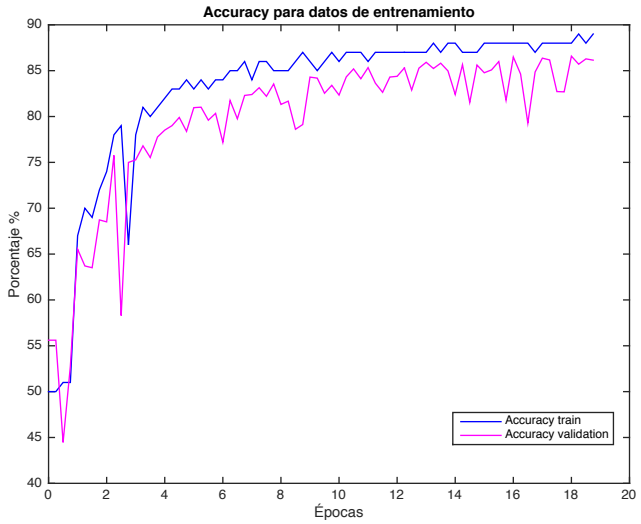


Figura 5-8: Curvas de *accuracy* para el Experimento 7

En la **Figura 5-8** se observa que se ha obtenido el mejor rendimiento en entrenamiento de entre todos los experimentos. Además, la curva de *accuracy* para el conjunto de validación también ha aumentado llegando a casi un 87%. En cuanto a los datos de test, el valor de *accuracy* obtenido ha sido de un 85,25%, el mayor hasta ahora, y un 21,9927% de EER. En EER no mejora pero está más cercano a los resultados de la DNN. Por último, en este caso el sistema ha acertado más con la clasificación de fragmentos de no voz que de voz, como se verifica en la **Tabla 5-13.**

		Clases reales	
		0	**1**
Clases predichas	**0**	43,94%	11,59%
	1	17,56%	26,90%

Tabla 5-13: Matriz de Confusión para el Experimento 7

6 Conclusiones y trabajo futuro

6.1 Conclusiones

Como conclusión general del proyecto se puede decir que se ha conseguido implementar un sistema de Detección de Actividad Voz basado en redes neuronales obteniendo precisiones de hasta un 85% en clasificación por tramas en el conjunto de test. Se ha partido de una DNN implementada con la librería de Python, Keras, y se han ido cambiando parámetros hasta llegar a los mejores resultados que se han podido conseguir. Además, esta red se ha comparado con otra LSTM para ver cuál trabaja mejor para la tarea de VAD.

Para concluir acerca de cuál ha sido el mejor, se han mostrado unas comparaciones y valoraciones de los valores más representativos del rendimiento de cada sistema: DNN, LSTM con optimizador RMSprop y LSTM con optimizador Adam.

En la **Tabla 5-2** del apartado de *Pruebas y Resultados* se puede observar una comparación del porcentaje EER entre los resultados obtenidos para los 7 experimentos principales realizados. Este valor, como se ha dicho, es el punto en el que las curvas de las probabilidades de falsa aceptación y falso rechazo coinciden. En la tabla se observa que la red con peor EER es la LSTM utilizando como optimizador Adam y número de duración de las secuencias 150 *frames*, mientras que el optimizador RMSprop genera una pequeña mejora en este valor. Se observa que en todos los experimentos de la red DNN, el ERR es mejor que en cualquiera de los experimentos de la LSTM. Esto quiere decir que las arquitecturas de DNN utilizadas separan mejor las dos clases sin tener en cuenta un umbral.

En el caso del *accuracy,* casi todos los experimentos se encuentran cerca del 80%. Sin embargo, en este caso también se ha obtenido que, en general, los experimentos que utilizan el modelo de la DNN tienen mayor precisión que los de la LSTM, salvo para el último caso. Dado que para una Detección de Actividad de Voz la información importante es la del intervalo de cambio de voz a no voz y viceversa, ya que son zonas críticas, al tener secuencias más largas se añade información innecesaria. Esto hace que se ensucie el sistema. Por ello, se ha decidido realizar un último experimento con menor número de *frames* de contexto para comprobar que, como se esperaba desde un principio, la red LSTM tuviera un mejor rendimiento y produjera mejores resultados.

Completados los distintos experimentos y teniendo en cuenta los resultados, se puede afirmar lo siguiente. La red neuronal recurrente, LSTM, de este Trabajo Fin de Grado proporciona mejores resultados en cuanto a *accuracy* para los conjuntos de datos utilizados. Aún así, para un modelo simple de DNN con dos capas ocultas genera un buen resultado también. Esto podría deberse a que no es estrictamente necesaria la información de memoria de las LSTM para saber si alguien está hablando o no.

6.2 Trabajo futuro

Existen muchos modelos de ANN para los que se puede implementar la tarea de Detección de Actividad de Voz. En este Trabajo Fin de Grado simplemente se han usado dos tipos, DNN y LSTM-RNN, por lo que quedan muchas arquitecturas que explorar e hiper-parámetros que variar.

En cuanto al tiempo de entrenamiento, se puede alargar mucho más para tener cierta idea de cómo puede variar la salida del sistema o el propio entrenamiento a lo largo del aprendizaje.

Por otra parte, se quiere recalcar que, a la hora de entrenar, se ha utilizado un método de la librería Keras al que se le pasan las características y las etiquetas por separado. Sin embargo, se trató de implementar otro método al que se le pasaba un generador, el cual tiene como entrada la lista de datos de training o de validación y él mismo los divide en características y etiquetas devolviendo ambos para cada *batch*, fijando este último valor. El método entrena el modelo ejecutando en paralelo el generador, por lo que el gasto de memoria es mucho menor y, obviamente, más rápido. Por tanto, sería una buena idea en cuanto a implementación del entrenamiento de la red para incorporar a un futuro caso de utilización de redes neuronales ya que es muy probable que, utilizando los parámetros adecuados, se obtengan mejoras en tiempo si es que se tiene una gran cantidad de datos.

Por último, debido a que a la salida del sistema se han clasificado erróneamente muchos fragmentos de no voz, sería interesante implementar una arquitectura que realizase una limpieza de la señal antes o después del entrenamiento. Aunque más que limpiar la señal en sí, se podría filtrar las etiquetas predichas para suavizar la salida.

Referencias

[1] M. H. Moattar and M. M. Homayounpour, "*A Simple But Efficient Real-Time Voice Activity Detection Algorithm*", en IEEE Xplore (agosto 2009)

[2] J. Ramírez, J. M. Górriz and J. C. Segura, "*Voice Activity Detection. Fundamentals and Speech Recognition System Robustness*", en IntechOpen (junio 2007)

[3] Md Sahidullah and Goutam Saha, "*Comparison of Speech Activity Detection Techniques for Speaker Recognition*", en Cornell University Library, pp. 1-2 (Octubre 2012)

[4] K. El-Maleh and P. Kabal, "*Comparison of voice activity detection algorithms for wireless personal communications systems*", en Electrical and Computer Engineering. Engineering Innovation: Voyage of Discovery. IEEE Canadian Conference, pp. 470-473 (mayo 1997)

[5] F. Beritelli, S. Casale, G. Ruggeri, and S. Serrano, "*Performance evaluation and comparison of G.729/AMR/fuzzy voice activity detectors*", en Signal Processing Letters, IEEE, vol. 9, no. 3, pp. 85-88 (marzo 2002)

[6] R. Venkatesha Prasad, A. Sangwan, H. Jamadagni, M. Chiranth, R. Sah, and V. Gaurav, "*Comparison of voice activity detection algorithms for voip*", en Computers and Communications. Proceedings. ISCC. Seventh International Symposium, pp. 530-535 (2002)

[7] J. Haigh and J. Mason, "*Robust voice activity detection using cepstral features*", en IEEE Region 10 Conference on Computer, Communication, Control and Power Engineering (TENCON'93), pp. 321-324 (octubre 1993)

[8] Md. Afzal Hossan, S. Memon and Mark A. Gregory, "*A Novel Approach for MFCC Feature Extraction*", en IEEE Xplore (febrero 2011)

[9] J. Ramírez, P. Yelamos, J.M. Gorriz and J.C. Segura, "*SVM-based speech endpoint detection using contextual speech features*", en Electronics Letters (abril 2006)

[10] Hongfei Ding, Koichi Yamamoto and Masami Akamine, "*Comparative Evaluation of Different Methods for Voice Activity Detection*" (2008)

[11] Juan Miguel Marín Diazaraque, "*Introducción a las redes neuronales aplicadas*", pp. 11-16 (2012)

[12] C. Lamas Álvarez, "*Mejora de voz mediante redes neuronales profundas*", pp. 24-25 (junio 2017)

[13] Robert Hecht-Nielsen, "Theory of Backpropagation Neural Networks", en Neural Networks for Perception, pp. 65-93 (1992)

[14] Lei Jimmy Ba and Rich Caruana, "*Do Deep Nets Really Need to be Deep?*", (diciembre 2013)

[15] Martin Sundermeyer, Ralf Schlu¨ter, and Hermann Ney, "*LSTM Neural Networks for Language Modeling*" (Septiembre 2012)

[16] http://colah.github.io/posts/2015-08-Understanding-LSTMs/

[17] Ruben Zazo, Tara N. Sainath, Gabor Simko and Carolina Parada, "*Feature Learning with Raw-Waveform CLDNNs for Voice Activity Detection*", (septiembre 2016)

[18] Javier Marcos Quirós, "*Extracción de características mediante Redes Neuronales Multicapa para Reconocimiento de Idioma*", (mayo 2017)

[19] Tuan V. Pham, Chien T. Tang and Michael Stadtschnitzer, "*Using Artificial Neural Networks for Robust Voice Activity Detection Under Adverse Conditions*", en IEEE Xplore (Julio 2009)

[20] Ivan J. Tashev and Seyedmahdad Mirsamadi, "*DNN-based Causal Voice Activity Detection*", en Information Theory and Applications Workshop (febrero 2016)

[21] Thad Hughes and Keir Mierle, "*Recurrent Neural Networks for Voice Activity Detection*", (mayo 2013)

[22] Florian Eyben, Felix Weninger, Stefano Squartini and Björn Schuller, "Real-Life Voice Activity Detection with LSTM Recurrent Neural Networks and an Application to Hollywood Movies", (octubre 2013)

[23] N.Ryant, M. Liberman and J. Yuan, "*Speech Activity Detection on Youtube Using Deep Neural Networks*", en Interspeech (2013)

[24] Stephen Cass, "*The 2017 Top Programming Languages*", en IEEE Spectrum (Julio 2017)

[25] Documentación Keras: https://keras.io
[26] Documentación Tensorflow: https://www.tensorflow.org
[27] Documentación Theano: http://deeplearning.net/software/theano/
[28] Documentación Matlab: https://es.mathworks.com/products/matlab.html
[29] Documentación cuDNN: https://developer.nvidia.com/cudnn
[30] https://www.nist.gov/itl/iad/mig/nist-2017-pilot-speech-analytic-technologies-evaluation
[31] A.R. Mohamed, G.E. Dahl anf G. Hinton, "*Deep belief networks for speech recognition*", en Proc. NIPS Workshop on Deep Learning for Speech Recognition and Related Applications, (2009)
[32] Zou, Weibao and Li, Yan and Tang, Arthur, "*Effects of the numer of hidden nodes used in a structured-based neural network on the reliability of image classification*", en Neural Computing and Applications (2009)

Glosario

VAD	Voice Activity Detection
ANN	Artificial Neural Network
DNN	Deep Neural Network
RNN	Recurrent Neural Network
LSTM	Long Short-Term Memory
CNN	Convolutional Neural Network
MFCC	Mel Frequency Cepstral Coefficients
OpenSAT	Open Speech Analytic Technologies
VAST	Video Annotation for Speech Technologies
PSC	Public Safety Communications
SSSF	Sofa Super Store Firefighters
LMR	Land Mobile Radio
KWS	Key Word Search
ASAR	Automatic Speech Recognition
CTS	Conversational Telephone Speech
IARPA	Intelligence Advanced Research Projects Activity
NIST	National Institute of Standards and Technologies
STE	Short Time Energy
STZCR	Short Time Zero Crossing Rate
STAM	Short Time Average Magnitude
MSE	Mean Squared Error
SNR	Signal to Noise Ratio
EER	Equal Error Rate
DT	Decision Tree
SVM	Support Vector Machines
GMM	Gaussian Mixture Models
ReLU	Rectifier Lineal Unit
Adam	Adaptive Moment Estimation
SGD	Stochastic Gradient Descent
GPU	Graphics Processing Unit
CPU	Central Processing Unit
TN	True Negatives
TP	True Positives
FN	False Negatives
FP	False Positives
FA	False Acceptance
FR	False Rejection
TFG	Trabajo Fin de Grado

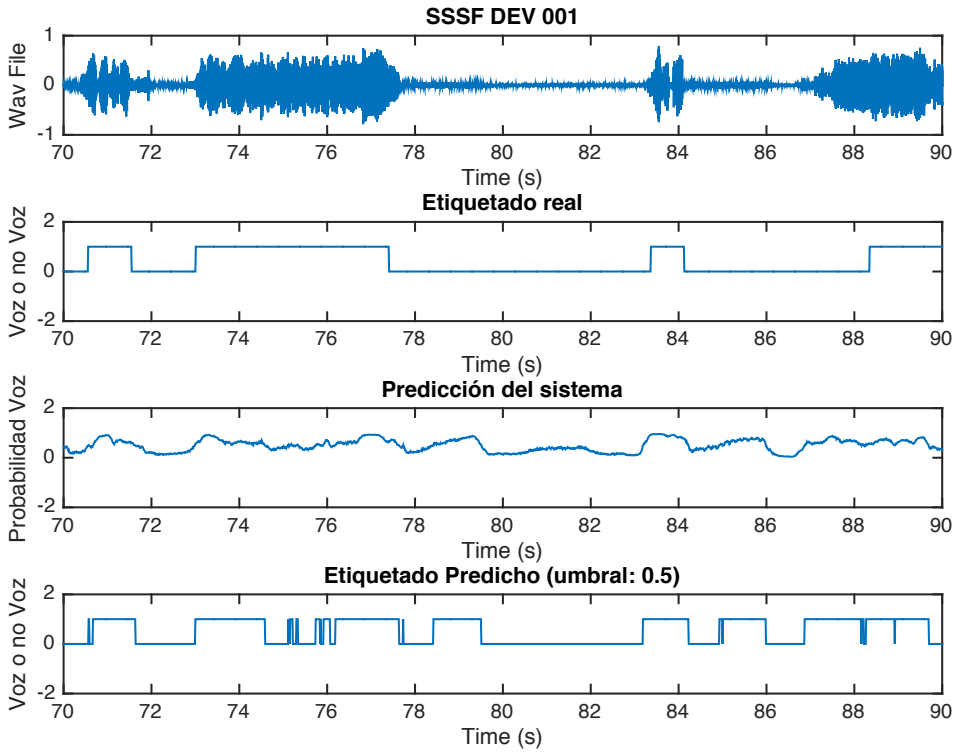
Anexo

A Representaciones Gráficas

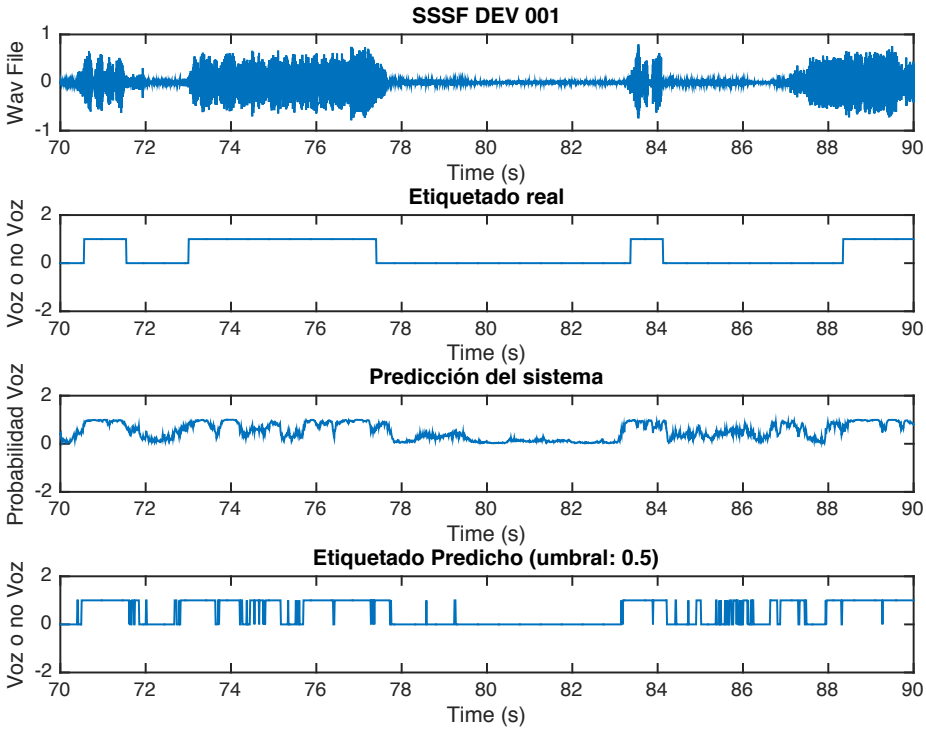
En este Anexo se añadirá a la memoria las distintas representaciones gráficas de la forma de onda de 20 segundos del primer archivo del conjunto de test, junto con su etiquetado real. A continuación, se le añadirá a la representación la salida del sistema en forma de probabilidades para cada clase y su etiquetado predicho en base a estas probabilidades.

- *Representación para el Experimento 2*
- *Representación para el Experimento 3*
- *Representación para el Experimento 4*
- *Representación para el Experimento 5*
- *Representación para el Experimento 6*
- *Representación para el Experimento 7*

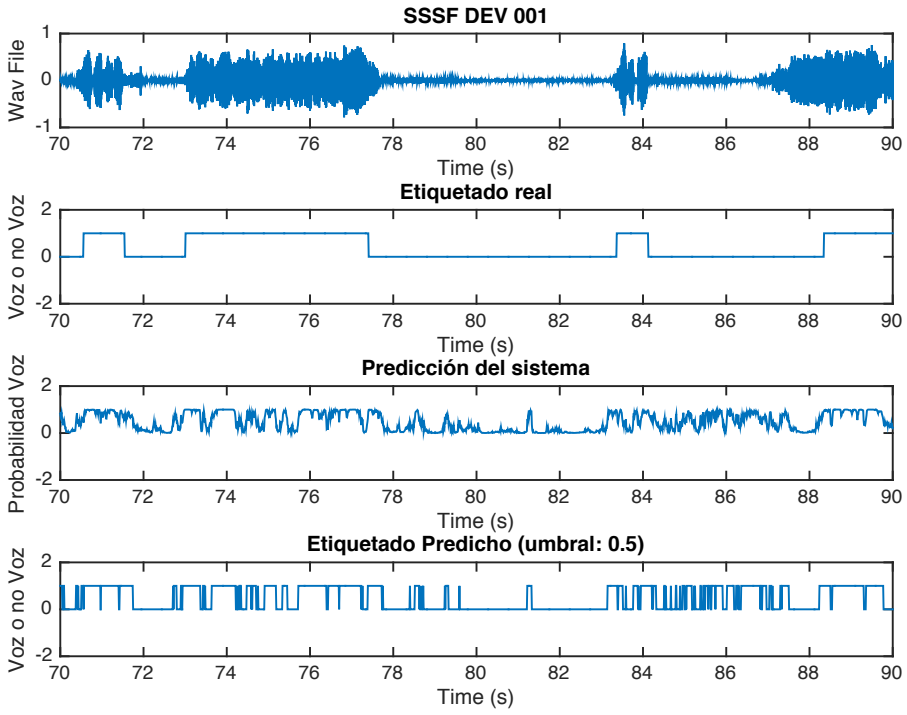
Forma de onda y etiquetado real comparados con predicciones del Experimento 2



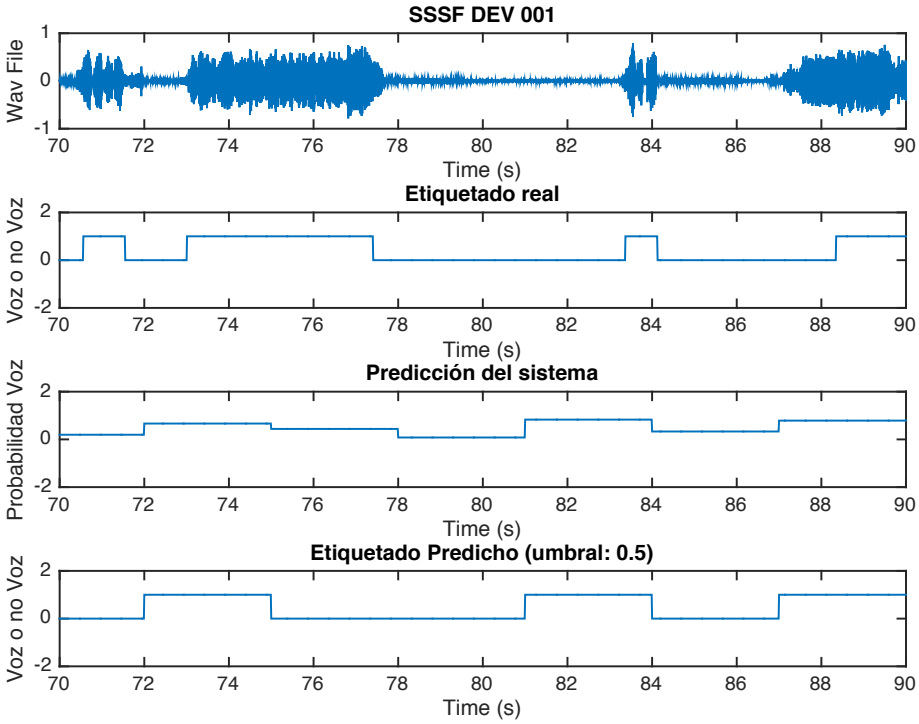
Forma de onda y etiquetado real comparados con predicciones del Experimento 3



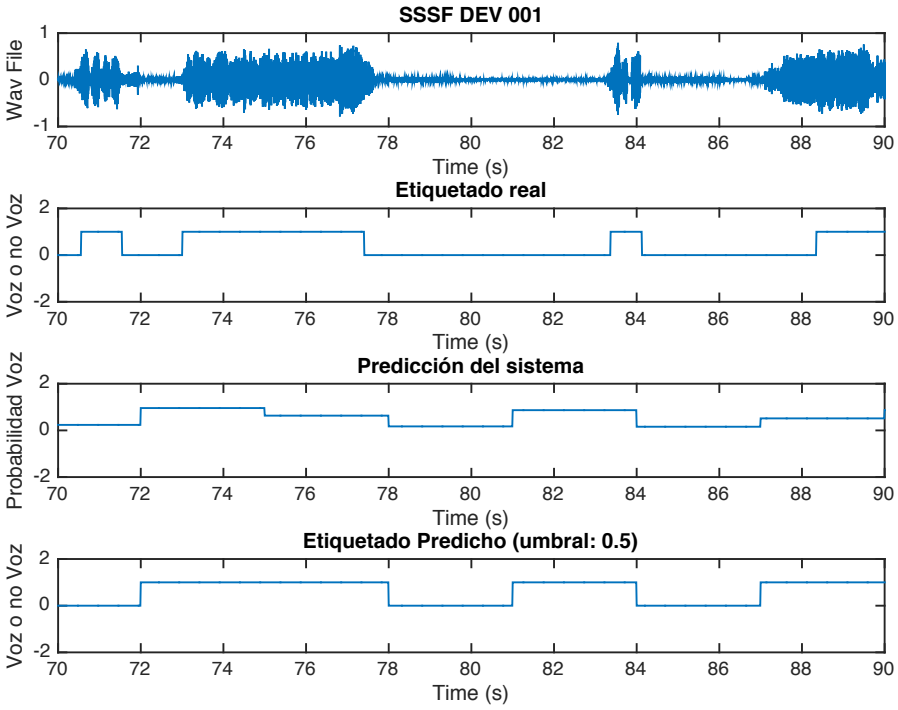
Forma de onda y etiquetado real comparados con predicciones del Experimento 4



Forma de onda y etiquetado real comparados con predicciones del Experimento 5



Forma de onda y etiquetado real comparados con predicciones del Experimento 6



Forma de onda y etiquetado real comparados con predicciones del Experimento 7