Universidad Autónoma de Madrid
Escuela politécnica superior

Grado en Ingeniería de Tecnologías y Servicios de Telecomunicación

Trabajo Fin de Grado

ETIQUETADO AUTOMÁTICO DE SEGMENTOS DE AUDIO EN DISTINTAS UNIDADES FONÉTICAS

Autor: Daniel Herreros Salamanca
Tutor: Alicia Lozano Díez
Ponente: Joaquín González Rodríguez

Junio 2016

ETIQUETADO AUTOMÁTICO DE SEGMENTOS DE AUDIO EN DISTINTAS UNIDADES FONÉTICAS

Autor: Daniel Herreros Salamanca
Tutor: Alicia Lozano Díez
Ponente: Joaquín González Rodríguez

Biometric Recognition Group - ATVS
Dpto. de Tecnología electrónica y de las comunicaciones
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Junio 2016

Resumen

Resumen

Este Trabajo Fin de Grado tiene como objetivo realizar el etiquetado de segmentos de audio en fonemas y trifonemas. Para ello realizamos experimentos basados en la extracción de coeficientes MFCC, las características Delta y Delta-Delta, SAT, MMI y fMMI. Finalmente se realizará el entrenamiento de distintas redes neuronales profundas para realizar el etiquetado.

Para los entrenamientos usaremos principalmente dos herramientas: Kaldi y Theano. Kaldi nos proporcionará herramientas para el entrenamiento de los sistemas anteriores a las redes neuronales, pero también nos permitirá entrenar una red neuronal. Por otra parte, las librerías de Theano nos permitirán entrenar otra serie de redes neuronales profundas mediante el uso de arquitecturas GPU.

Las redes neuronales profundas (DNN), brindan mejores resultados en el etiquetado de voz que el resto de experimentos realizados durante el desarrollo del Trabajo Fin de Grado, por ello nos centramos en estudiar sus resultados. Realizaremos comparaciones entre los resultados obtenidos por las DNNs y los resultados de los experimentos nombrados anteriormente. De la misma manera compararemos los distintos resultados de las DNNs entre ellos, teniendo en cuenta el tiempo de entrenamiento y la precisión en cuanto palabras o ventanas acertadas.

Como base de datos se usará la base de datos Switchboard de LDC, con número de serie LDC97S62. Dicha base de datos consta de alrededor de 2400 conversaciones entre dos locutores en inglés. La base de datos se dividirá en distintos subconjuntos para el entrenamiento y validación de los sistemas entrenados.

Finalmente, hablaremos sobre las líneas de investigación futuras de las DNNs para el etiquetado de segmentos de audio como son las redes convolucionales y recurrentes.

Palabras Clave

Reconocimiento del habla, etiquetado automático, aprendizaje profundo, parametrización de voz, redes neuronales, Kaldi, Theano, GPU.

Abstract

In this undergraduate project we aim to create a system capable of labelling audio segments in phones and triphones. In order to achieve our objective, we are going to use systems based on MFCC extraction, delta and delta-delta features, SAT, MMI and fMMI. Finally we are going to train a set of deep neural networks to approach the labelling problem.

Our main tools to create the systems and train them are: Kaldi and Theano. Kaldi will give us the tools to train the systems previous to the implementation of the deep neural networks, but it will also give us the possibility to train a net by using its tools. Theano libraries will give us access to training deep neural networks using GPU architectures.

Deep neural networks (DNN) have bring better results to the labeling voice than the rest of systems developed in this undergraduate project, due to this we are going to focus on their results. Moreover, we are going to compare the results of the DNNs with the results of the rest of systems. In the same vein, we are going to compare the results of the differents DNN between them, having in mind their training time and accuracy over words or frames.

As database we are going to use the LDC's database with serial number LDC97S62. This database is compromised of around 2400 two sided telephone conversations in English. The database is going to be divided in different segments, in order to have a training and test set for the systems.

Finally, we are going to discuss the future research lines based on DNNs for audio segments labelling, like convolutional nets and recurrent nets.

Key words

Speech recognition, automatic labeling, deep learning, voice parametrization, neuronal netowrks, Kaldi, Theano, GPU.

Agradecimientos

"That is not dead which can eternal lie.
And with strange aeons even death may die."

The Nameless City - H.P. Lovecraft

Primero, debo agradecer a mi abuelo Rodolfo, por darme los medios económicos para poder estudiar esta carrera. También a mis dos hermanos, Demián y Diego, por que a pesar de las peleillas de hermanos, siempre han intentado mantenerme animado durante estos años. A mis padres por ofrecerme un lugar donde vivir y haber inmigrado a este país para mantener mi seguridad y educación.

Luego he de agradecer a mis amigos, a los nacionales y a los internacionales. A los del BetaMax: Abel, Juan y Nelson, con quienes tantas cosas he vivido y que han sido un apoyo fundamental en mi vida. Gracias a los otros dos tercios del triunvirato Europeo: Jonas y Bálint, Thank you guys. Y como no, a Javi, que es quien más tiempo ha tenido que aguantarme en el mundo de las ingenierías, y a quien debo cientos de paquetes de chicles y alguna que otra cerveza a precio de oro.

Agradezco a los compañeros de ATVS, por permitirme desarrollar este Trabajo Fin de Grado en sus laboratorios, pero sobretodo a Joaquín y Alicia, por su paciencia y brindarme la oportunidad de realizar este proyecto.

Finalmente y no por ello menos importante, quiero agradecer a tres profesores de mi infancia: Julián, Suescún y Jorge, fueron ellos los que sentaron las bases de mi cabeza y me ayudaron a madurar lo suficiente como para entender el valor de mis estudios.

También agradezco a todo el que me haya brindado su apoyo a lo largo de este viaje, y quien haya olvidado nombrar en estos agradecimientos, que me disculpe, tengo un TFG que defender.

Índice general

Índice de Figuras IX

Índice de Tablas X

1. Introducción 1

1.1. Motivación del proyecto . . . 1

1.2. Objetivos y enfoque . . . 2

1.3. Metodología y plan de trabajo . . . 2

2. Etiquetado automático de audio. Estado del arte 3

2.1. Introducción . . . 3

2.2. Etiquetado automático de segmentos de audio . . . 4

2.2.1. Introducción . . . 4

2.2.2. Sistemas de etiquetado tradicionales . . . 5

2.3. Etiquetado mediante DNN's . . . 5

2.3.1. Introducción . . . 5

2.3.2. Uso de las DNN's dentro de la detección y etiquetado de voz . . . 7

2.4. Métodos de computación . . . 7

2.4.1. Introducción . . . 7

2.4.2. CPU . . . 8

2.4.3. GPU . . . 8

2.4.4. Comparación entre CPU y GPU . . . 8

2.5. Herramientas de tratamiento de señal y cálculo . . . 9

2.5.1. Kaldi . . . 9

2.5.2. Theano . . . 9

3. Bases de datos 11

3.1. Switchboard-1 Release 2 . . . 11

4. Diseño y desarrollo 13

4.1. Preparación de la base de datos . . . 13

4.2. Parametrización y segmentación de la base de datos 14
4.3. Sistema de entrenamiento para Kaldi 14
4.4. Evaluación de los resultados de Kaldi 15
4.5. Sistema de entrenamiento para Theano 16
4.6. Evaluación de los resultados de Theano 17

5. Experimentos y Resultados **19**
5.1. Sistema basado en fonemas usando alineamiento de MFCC 19
5.2. Sistema basado en trifonemas usando información de deltas 19
5.3. Sistema basado en trifonemas usando LDA y MLLT 20
5.4. Sistema basado en trifonemas usando SAT 21
5.5. Sistema basado en trifonemas usando MMI 21
5.6. Sistema basado en trifonemas usando fMMI 22
5.7. Comparación de los resultados previos a las DNNs 23
5.8. Sistema basado en la DNN de Kaldi 24
5.9. Sistema basado en DNNs de Theano 26
5.10. Comparación de todos los resultados 28

6. Conclusiones y trabajo futuro **31**

Glosario de acrónimos **33**

Bibliografía **34**

A. Entorno de trabajo **37**

B. Manual del programador **39**

Índice de Figuras

1.1. Interacción humano maquina mediante la voz. 1

2.1. Aparato fonador y espectros de señales vocales [1]. 4
2.2. Función sigmoide entre -5 y 5 6
2.3. Estructura simplificada de una DNN 7

4.1. Ciclo prototipo 14
4.2. Flujo de información 16

5.1. Resultados MMI 22
5.2. Resultados fMMI 23
5.3. Resultados sin redes neuronales 23
5.4. Tangente hiperbólica entre -2 y 2 24
5.5. Resultados de los experimentos de Kaldi incluyendo la DNN 28
5.6. Resultados de las redes de Theano acompañadas de número de datos de entrada y clases de salida 28
5.7. Evaluaciones por época de la red de Theano entrenada con los alineamientos de fMMI 29

Índice de Tablas

5.1. Resultados entrenamiento para fonemas usando sólo los MFCC 19
5.2. Resultados entrenamiento mediante delta y delta-delta 20
5.3. Resultados entrenamiento mediante LDA+MLLT 20
5.4. Resultados entrenamiento mediante LDA+MLLT+SAT usando el conjunto de 100000 locuciones y el conjunto de entrenamiento en su totalidad 21
5.5. Resultados del entrenamiento MMI para el conjunto de 100000 locuciones 21
5.6. Resultados del entrenamiento MMI para el conjunto de entrenamiento en su totalidad 22
5.7. Resultados del entrenamiento fMMI para el conjunto de entrenamiento de 100000 locuciones 22
5.8. Resultados del entrenamiento fMMI para el conjunto de entrenamiento en su totalidad 23
5.9. Resultado de la DNN entrenada usando Kaldi 26
5.10. Resultados de los entrenamientos de redes neuronales usando las librerías de Theano 27
5.11. Comparación GPU vs. CPU para el entrenamiento de la red basada en los alineamientos de tri4a fMMI 27

1 Introducción

1.1. Motivación del proyecto

La motivación de este Trabajo Fin de Grado es conseguir explorar los sistemas de reconocimiento de voz. Primero debemos destacar lo importante que son las tecnologías de reconocimiento del habla basadas en las tecnologías que utilizan los sistemas del estado del arte de en este campo. El reconocimiento del habla no es sólo importante para el desarrollo de sistemas que mejoren la interacción con el usuario, en la actual y evolutiva sociedad digital y de la información, sino que también permite estrechar barreras en contra de las discapacidades y los diferentes idiomas.

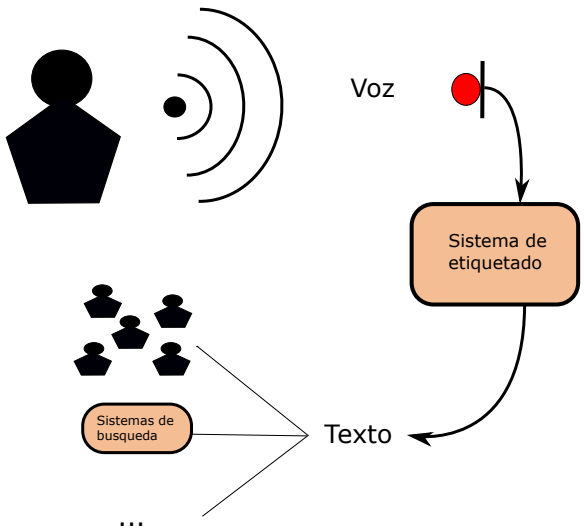

Figura 1.1: Interacción humano maquina mediante la voz.

Si nos centramos en las tecnologías, la principal motivación de este Trabajo Fin de Grado es dar a conocer las redes neuronales profundas, a las que nos referiremos por sus siglas en inglés DNN (deep neural networks). Intentaremos exponer las DNN como posible solución al problema del etiquetado de segmentos voz en distintas unidades fonéticas. Finalmente expondremos

los resultados y conclusiones sobre los sistemas estudiados, además de explorar las líneas de investigación futura sobre estos sistemas.

1.2. Objetivos y enfoque

Como hemos expuesto en la motivación del Trabajo Fin de Grado, estudiaremos el uso de las DNNs para el reconocimiento de voz, por tanto, nuestro objetivo es obtener resultados sobre el etiquetado de segmentos de audio de una base de datos y comprobar si las DNNs presentan realmente una mejora sobre los métodos tradicionalmente usados en estos ámbitos.

Para demostrar si las redes neuronales profundas brindan mejores resultados en el etiquetado de voz, realizaremos una serie de experimentos previos. Los resultados de estos experimentos previos, serán comparados con los resultados de las DNNs.

Mantendremos un enfoque empírico, es decir, buscaremos obtener resultados numéricos para cada sistema desarrollado. Finalmente se presentaran las mejoras sobre las DNNs que se pueden usar para el reconocimiento de voz y el etiquetado de este tipo de señales.

1.3. Metodología y plan de trabajo

A lo largo del Trabajo Fin de Grado usaremos la siguiente metodología: Empezaremos realizando experimentos donde etiquetamos parte de la base de datos en fonemas, los resultados de estos experimentos serán usados como base para nuevos experimentos usando distintas técnicas para el etiquetado de trifonemas. Posteriormente, se entrenarán redes neuronales profundas usando dos sistemas distintos y se compararán los resultados obtenidos entre ellos con los resultados obtenidos sin el uso de DNNs. Finalmente se compararán los resultados obtenidos de cada DNN entre ellos.

2

Etiquetado automático de audio. Estado del arte

2.1. Introducción

Como hemos expuesto en la motivación, el reconocimiento de la señal hablada es útil no sólo en aplicaciones como podría ser, el reconocimiento de comandos de voz para un sistema de control. El tratamiento de la señal de voz y su correcto etiquetado cubren un abanico de utilidades mucho mayor. Esto se debe a que la voz es el principal método de comunicación entre los seres humanos

Para entender cómo funcionan los sistemas de reconocimiento de voz primero explicaremos a grandes rasgos cómo se genera la señal de voz. Además expondremos cuáles son las componentes básicas de la voz.

Empezaremos por descomponer el aparato fonador en sus distintas partes, explicar en qué consisten cada una y cuáles son sus funciones en la producción de la señal de voz.

El sistema fonador humano se divide en tres partes: el tracto subglotal, las cavidades del tracto vocal y las cavidades del tracto nasal y paranasal.

La voz se genera mediante un flujo de aire originado en los pulmones gracias a la compresión de estos por un músculo llamado diafragma. A este flujo de aire lo podemos considerar una señal impulsiva que excita el tracto vocal y nasal como si de un filtro se tratara, generando una respuesta en frecuencia que representará el fonema generado en cada instante [2]. En la figura 2.1, extraída de *Springer Handbook of Acoustics* (Springer 2007, p.682), podemos analizar las distintas partes del aparato fonador, junto con las cuerdas vocales (*Vocal folds*). Las gráficas a la derecha muestran en orden descendente: el espectro de la señal radiada, la envolvente del espectro (los formantes), el espectro de la excitación glotal y la forma de onda producida por la corriente de aire expulsado a través de la glotis.

Un fonema es la unidad mínima de sonido inteligible que produce la señal de voz, existen un total de 107 fonemas básicos, 52 signos diacríticos y 4 prosodías en el alfabeto fonético internacional [3] pero cada idioma usa un subconjunto determinado de estos símbolos, por lo que dentro de un idioma sólo se trabaja bajo una variabilidad de fonemas determinada. Siguiendo las normas dadas por el alfabeto fonético internacional, para referirnos a un fonema lo escribimos entre barras oblicuas, i.e. /ä/, el cual se refiere al fonema asociado a la vocal 'a' del español.

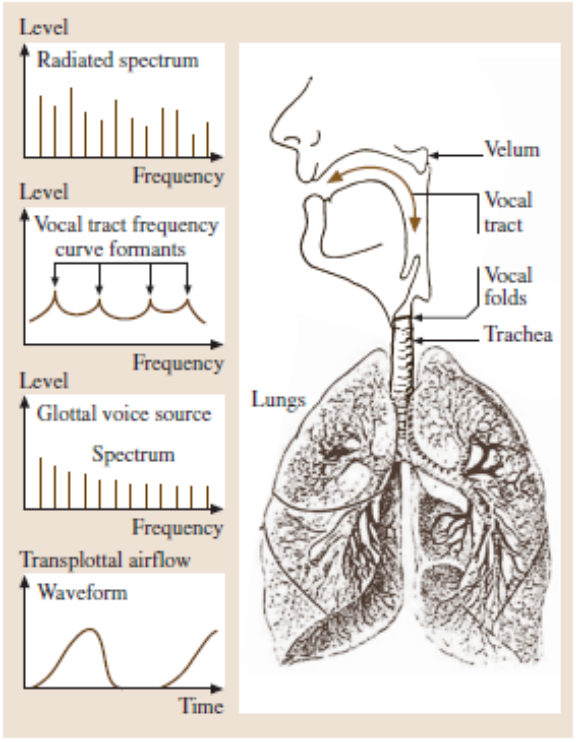

Figura 2.1: Aparato fonador y espectros de señales vocales [1].

Un trifonema consiste en un conjunto de tres fonemas. Por la cantidad de combinaciones posibles y la naturaleza de los sonidos fonéticos, no todas las combinaciones de fonemas son posibles, además estos conjuntos varían para cada idioma. Por otra parte, los trifonemas brindan más información estadística que los fonemas por si solos, hecho que puede ser explotado por los algoritmos de *Machine Learning*, como veremos con los experimentos de este Trabajo Fin de Grado.

2.2. Etiquetado automático de segmentos de audio

2.2.1. Introducción

El etiquetado de segmentos de audio, consiste en transcribir la forma de onda de dicho segmento en algo que sea inteligible para el ser humano. Un ejemplo es el uso de etiquetado de voz en sistemas de control de manos libres para un vehículo.

El trabajo de etiquetado de voz es una labor tediosa, por lo que se ha intentado automatizar desde hace décadas. En este Trabajo Fin de Grado se trabajará con sistemas de reconocimiento de voz que sean capaces de transcribir, automáticamente, la forma de onda de una señal de voz, en las palabras y sonidos que la representan. Para realizar la tarea de etiquetado de forma automática el sistema ha de entender qué representa cada segmento de la forma de onda. Para realizar este proceso, se ha recurrido a distintos algoritmos de *Machine Learning* con los que se han intentado realizar procesos de etiquetado con distintos grados de complejidad y distintos resultados. En el próximo apartado hablaremos de los distintos métodos usados históricamente para el etiquetado de voz, de los cuales algunos formarán parte de los experimentos realizados durante este Trabajo Fin de Grado.

2.2.2. Sistemas de etiquetado tradicionales

Algunos de los métodos de reconocimiento y etiquetado de la señal de voz son:

- Alineamiento dinámico temporal: También llamado DTW, por sus siglas del inglés *Dynamic Time Warping* [4]. DTW consiste en alinear los coeficientes de una señal test con una señal modelo, la cual suele provenir de una señal de referencia extraída de un entrenamiento previo. DTW ha caído en desuso en favor de otros sistemas de etiquetado, por no ser demasiado robusto en contra de la naturaleza variable de la señal de voz y del canal de captación, lo que le reduce su utilidad en entornos sin condiciones controladas.

- Entrenamiento mediante coeficientes, velocidades y aceleraciones: Este método consiste en buscar coeficientes que representen la señal y compararlos con los de un modelo entrenado anteriormente, sin ajustarlo temporalmente como con DTW, pero usando la primera y segunda derivada de la serie temporal que forman los coeficientes para aumentar la precisión de los resultados. Mediante este método, aunque separado en dos partes distintas, se entrenarán modelos que forman parte de los experimentos de este Trabajo Fin de Grado. A la utilización de velocidades y aceleraciones se le suele llamar delta y delta-delta.

- Análisis discriminante lineal: Esta técnica consiste en buscar combinaciones lineales que sean capaces de caracterizar las distintas clases en las que queremos representar la señal, en nuestro caso sería en las distintas unidades fonéticas, sean fonemas o trifonemas. En este Trabajo Fin de Grado usaremos LDA (Linear Discriminant Analysis) junto a MLLT (Maximum Likelihood Linear Transform) para realizar uno de los experimentos de entrenamiento.

- Modelos ocultos de Markov: Estos modelos forman cadenas mediante el uso de los estadísticos de la señal. El resultado es la caracterización de los distintos estados de un segmento, por ejemplo los distintos fonemas que forman una palabra e inclusive las distintas etapas dentro de la pronunciación de un fonema. Los HMM (Hidden Markov Model) sustituyeron a los algoritmos DTW en casi todas los sistemas de reconocimiento de voz en la década de los 90 [5]. Un HMM se define por su número de estados, número de símbolos observables, la matriz de probabilidades de transición, la distribución de probabilidad en cada estado y la probabilidad inicial de posicionarse en un estado dado.

- Modelos de mezclas Gaussianas: Comúnmente generados a partir de la cuantificación vectorial de los datos, los GMMs consisten de un grupo de funciones de probabilidad Gaussianas, las cuales discriminan los datos en un número dado de clases. La función de distribución que caracteriza a cada GMM se genera a partir de la suma ponderada de las probabilidades de las distribuciones Gaussianas que lo forman, por esto se les denomina mezcla de Gaussianas. Mediante el uso de GMMs también se generan los llamados UBM (Universal Background Models), modelos de mezclas Gaussianas que representan un modelo de locutor genérico, los cuales nos permiten ser más robustos en contra de la falta de datos sobre nuevos hablantes, ya que generamos GMMs a partir de la adaptación del UBM [6].

2.3. Etiquetado mediante DNN's

2.3.1. Introducción

Las redes neuronales profundas surgieron como un método de clasificación basado en el funcionamiento del cerebro. La unidad mínima de computación de una red neuronal es una neurona.

La neurona de una red neuronal esta compuesta por una función matemática y una serie de conexiones de entrada y salida, con sus respectivos pesos. La función matemática que define una neurona suele ser una función no lineal, tal que permita discriminar de forma sencilla las distintas entradas. Comúnmente se han usado la función tangente hiperbólica y la función sigmoide. La función sigmoide es definida matemáticamente según la fórmula 2.1 y su representación gráfica es la de la figura 2.2.

$$S(t) = \frac{1}{1 + e^{-t}} \tag{2.1}$$

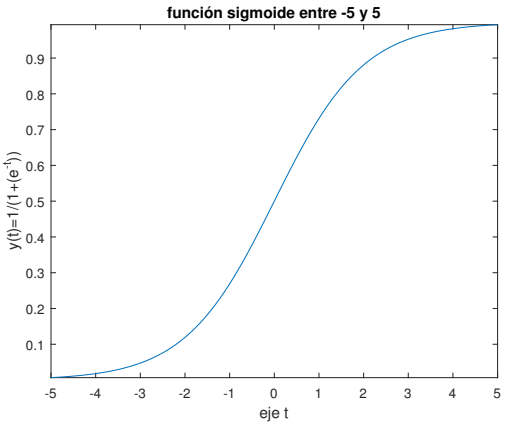

Figura 2.2: Función sigmoide entre -5 y 5

La función tangente hiperbólica se usará para entrenar uno de los sistemas desarrollados durante este Trabajo Fin de Grado, por lo que se definirá en profundidad más adelante.

Mediante los pesos de las conexiones de entrada y de salida obtenemos las funciones de coste, las cuales definirán como se ajusta la red al problema en cuestión. Los distintos métodos para estimar los pesos se expondrán posteriormente en este apartado.

Las neuronas se distribuyen formando lo que llamamos capas, el número de capas de una red neuronal se elige al diseñarla junto al número de neuronas por capa. Cada capa se encarga de analizar una parte de la información de entrada más abstracta. Usando como ejemplo una red dedicada a analizar imágenes, la primera capa trabajaría con la información a nivel pixel, detectando bordes y esquinas, la siguiente sería capaz de analizar conjuntos de bordes y esquinas que formen figuras, la próxima capa analizaría figuras y así sucesivamente, buscando cada vez un nivel más abstracto. La estructura básica de una DNN se puede observar en la figura 2.3.

Los pesos de las neuronas varían por cada iteración de entrenamiento buscando minimizar la función de coste global, para así minimizar el error de clasificación. Para ajustar los pesos usaremos algoritmos de descenso de gradiente, este tipo de funciones nos permiten converger hacia mínimos en la función de coste. En conjunto con el algoritmo de descenso por gradiente, se usan algoritmos de propagación de error, el más común es *Back Propagation*. El algoritmo de *Back Propagation*, consiste en dos etapas: En la primera etapa se comprueba el error entre una entrada a la red y su salida. En la segunda etapa el error se propaga de salida a entrada, capa a capa. El error es dividido entre las aportaciones de cada neurona y esta información se usa para ajustar los pesos y así minimizar la probabilidad de error.

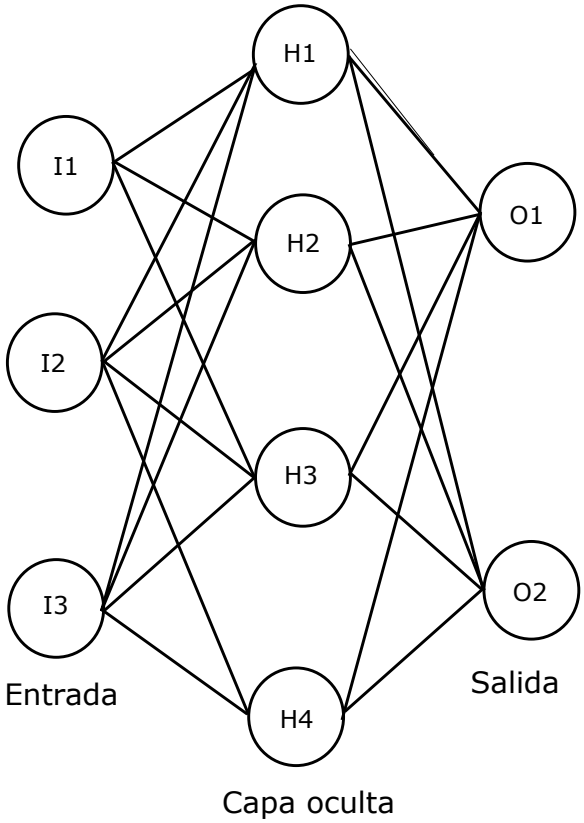

Figura 2.3: Estructura simplificada de una DNN

2.3.2. Uso de las DNN's dentro de la detección y etiquetado de voz

El uso de redes neuronales en sistemas de tratamiento la señal de voz como entrada está muy extendido, algunas de sus aplicaciones son: reconocimiento de idioma, reconocimiento de locutor, y detección voz/no voz. En una red neuronal diseñada para el análisis de voz no se suelen usar como entrada la forma de onda, si no que usan coeficientes que la representen de forma lo más aproximada posible, pero a una tasa de información mucho menor. Para obtener estos coeficientes, la forma de onda se separa en distintas ventanas, por cada ventana obtendremos un vector de coeficientes determinado. Este proceso, llamado parametrización de la señal de voz, será explicado en profundidad durante la memoria del desarrollo de los experimentos del trabajo Fin de Grado.

2.4. Métodos de computación

2.4.1. Introducción

Uno de los principales problemas y razón por la que las redes neuronales profundas se consideraron poco viables durante tanto tiempo, es la cantidad de recursos que utilizan. A nivel memoria, el problema se soluciona con memorias de mayor tamaño, esta solución fue alcanzada rápidamente por el incremento de la capacidad de las memorias en los últimos 30 años. El otro recurso que las redes neuronales necesitan en gran cantidad es tiempo de computación, debido al número de operaciones realizadas. En los últimos años este problema se ha ido solucionando debido a la mayor eficiencia de los sistemas de computación CPU y recientemente gracias a la computación mediante sistemas GPU.

2.4.2. CPU

CPU (Central Processing Unit) es la arquitectura de computación clásica de los ordenadores y servidores, surgida en los años 40 para alcanzar el nivel de los microprocesadores en la década de los 70 del siglo pasado. Las CPUs están diseñadas para realizar múltiples tareas ya que son la base de cualquier ordenador, también deben ser capaces de agilizar interrupciones y comunicaciones con periféricos entre otras operaciones, por lo que son elementos muy complejos pero con funciones muy generales.

Gracias a la aparición de las CPUs multinúcleos y multihilos, junto con la adaptación de los algoritmos a la computación en paralelo, el problema del tiempo de computo de las redes neuronales ha ido desapareciendo. Estas tecnologías han permitido reducir el tiempo de computo de cada iteración de entrenamiento en gran medida.

2.4.3. GPU

Por otra parte GPU (Graphical Proccesing Unit) es una tecnología emergente en los sistemas *deep learning* pero que existe también desde hace muchos años. Principalmente usado en, como su propio nombre indica, procesamiento de imagen y vídeo, propulsado por la creciente demanda en generación de gráficos de calidad en el mundo del diseño, cinematografía y videojuegos.

Las GPU están diseñadas para realizar tareas en paralelo y usar datos de entrada en forma de vectores y matrices, ya que están diseñadas para trabajar con múltiples imágenes y formas al mismo tiempo. Esta parte de su diseño las hace increíblemente eficientes a la hora de trabajar en sistemas *deep learning*, ya que debido a las características de las redes neuronales profundas, poder paralelizar los cálculos y operar ágilmente con vectores y matrices minimiza el tiempo de computo de cada iteración.

Actualmente existen dos lenguajes principales que permiten explotar las posibilidades de paralelización de las GPUs, CUDA®, propietario del fabricante NVIDIA©, y la alternativa de código abierto OpenCL [7]. Por otra parte CUDA® se complementa con cuDNN, una librería que contiene funciones diseñadas para agilizar el proceso de ciertas partes de las redes neuronales como son los algoritmos de *back propagation*, convoluciones, normalización y activación de capas [8].

2.4.4. Comparación entre CPU y GPU

Si empezamos por las diferencias a nivel de coste, un CPU de 4 núcleos comercial de alta gama cuesta aproximadamente tres cuartas partes de lo que cuesta una GPU de la misma gama. Por otra parte las CPUs experimentales son capaces de paginar hasta 1 Terabyte de memoria RAM, mientras que las GPUs experimentales solo alcanzan 24 Gigabytes de memoria RAM. A la hora de trabajar con gran cantidad de información, una CPU es teóricamente más rápida por tener más datos en memoria RAM, esto significa menos accesos a disco para obtenerlos, pero las GPUs tienen un mayor ancho de banda para acceso a su memoria RAM, lo que les permite mover más datos que una CPU en la misma cantidad de tiempo. Un fallo secundario de las GPUs es que los datos los recibe por un puerto de la placa base, actualmente el más común es un puerto tipo PCI-E, el cual es más lento que la conexión entre la memoria RAM de la GPU y dicha GPU. Incluso con las ventajas que presentan la CPUs sobre las GPUs, estas han demostrado ser mucho más eficientes en el entrenamiento de redes neuronales profundas. Esta diferencia se debe a su alto poder de paralelización y agilidad de computación matricial, como se demostrará con algunos de los experimentos de este Trabajo Fin de Grado.

2.5. Herramientas de tratamiento de señal y cálculo

2.5.1. Kaldi

Kaldi es la principal herramienta que usaremos durante este Trabajo Fin de Grado, es una herramienta diseñada en C++ bajo una licencia Apache 2.0, su desarrollo se inicio en el año 2009 en la Universidad Johns Hopkins intentando conseguir una herramienta de bajo coste de desarrollo pero de alta calidad [9].

Kaldi, al ser una herramienta de código abierto y gratuita, ha permitido que en los últimos años la experimentación y desarrollo de nuevos sistemas de tratamiento de señales de audio [10].

Dentro de Kaldi, usaremos los scripts relacionados con los ejemplos de Kaldi para los entrenamientos de distintos sistemas para la base de datos *Switchboard*.

2.5.2. Theano

Theano es una librería de Python que permite agilizar la definición y operación de matrices de múltiples dimensiones. Las librerías de Theano serán usadas en la última parte de este Trabajo Fin de Grado para entrenar redes neuronales profundas. Las características más interesantes para el entrenamiento de redes neuronales profundas que nos ofrece Theano son:

- Uso transparente de las GPUs: Como hemos comentado en el apartado anterior el uso de GPUs reduce en gran medida el tiempo de computo necesario para entrenar las redes neuronales. Según Theano, el uso de sus librerías para el cálculo sobre GPU es hasta 140 veces más rápido que los cálculos en CPUs [11].
- Integración completa de los paquetes NumPy: El paquete para Python NumPy, permite el uso de un objeto de programación multidimensional, muy útil para los cálculos necesarios en el entrenamiento y evaluación de las redes neuronales.

3

Bases de datos

Para este Trabajo Fin de Grado hemos decidido usar una base de datos con un gran número de segmentos de audio, para poder segmentarla en distintas partes que podremos usar para entrenar y evaluar los modelos entrenados. Por otra parte el gran tamaño de la base de datos nos permitirá hacer entrenamientos con distintas cantidades de datos y ver cómo esto afecta a los resultados. Por ello hemos decidido usar *Switchboard-1 Release 2* [12] como base de datos en este Trabajo Fin de Grado.

3.1. Switchboard-1 Release 2

Esta base de datos fue recolectada por la empresa Americana Texas Instruments© bajo el patrocinio de DARPA©. La base de datos consta de alrededor de 2400 llamadas telefónicas en inglés, entre dos hablantes. La base de datos contiene un total de 543 locutores y por tanto voces distintas, con una distribución casi equitativa entre voces masculinas y femeninas (56 % de voces masculinas frente al 44 % de voces femeninas). Actualmente, esta base de datos está controlada por el *Linguistic Data Consortium* de la Universidad de Pensilvania, el cual le ha asignado el número de serie LDC97S62.

La base de datos se generó mediante el uso de un operador digital, que conectaba a dos usuarios y les daba un tema del cual deberían hablar (se usaron más de 70 temas distintos). Se establecieron dos condiciones, que se debían cumplir en el proceso de elección de locutores: que nunca coincidieran dos locutores más de una vez y que ningún locutor hablara dos veces sobre el mismo tema.

Como aspectos técnicos, cabe decir que debido a que el canal del cual provienen las pistas de audio que forman la base de datos es un canal telefónico, las pistas generadas tienen un ancho de banda de 8 Kiloherzios.

Desde su despliegue en 1997, las transcripciones de las pistas de audio han sido analizadas y corregidas por el *Institute for Signal and Information Processing.*

4

Diseño y desarrollo

4.1. Preparación de la base de datos

El primer paso que realizamos es preparar la extensa base de datos para los experimentos a realizar. Para ello el script principal de esta sección del Trabajo Fin de Grado realizada en Kaldi, run.sh, hace uso de los siguientes scripts de kaldi:

- swbd1_data_prep.sh: Este script es el encargado de descargar las transcripciones si no están ya en el sistema. A partir de las transcripciones, creará los segmentos correspondientes a dichas transcripciones siguiendo el siguiente esquema:

  `sw02001-A_000098-001156 sw02001-A 0.98 11.56`

  Donde la primera parte de la cadena es el segmento de la locución, la segunda el identificador del locutor, y la tercera y cuarta corresponden al tiempo de inicio y final de la locución. Además este script es el encargado de crear las listas de archivos y ordenar los archivos sphere (.sph).

- swbd1_prepare_dict.sh: Con este script eliminamos las palabras incompletas u otros tipos de anomalías del diccionario descargado y de las transcripciones.

- prepare_lang.sh: prepara la carpeta data/lang, un directorio que será usado a través de todos los experimentos de Kaldi. Este directorio contiene información sobre los fonemas y palabras obtenidos de las transcripciones, así como información de qué fonemas forman cada palabra.

- swbd_p1_train_lms.sh: Este script es el encargado de crear el *modelo del lenguaje*. Un modelo de lenguaje es una distribución de probabilidad sobre unidades lingüísticas. Lo que se busca al generar un modelo de lenguaje es ver cuál es la probabilidad de que una unidad lingüística vaya seguida de otra, para así generar la distribución de probabilidad. En nuestro caso recogemos las palabras de las transcripciones de tres en tres, generando un modelo *tri-grams*.

- format_lm.sh: Este script se encarga de cambiar el formato del modelo de lenguaje de *ARPA* a *Fst*. Debemos realizar este cambio ya que el resto de los scripts de Kaldi trabajan con modelos de lenguaje en formato *Fst*.

4.2. Parametrización y segmentación de la base de datos

Antes de iniciar los entrenamientos de los distintos sistemas de etiquetado procederemos a segmentar la base de datos. El proceso de segmentación se realizará utilizando las herramientas de Kaldi. Para empezar, parametrizaremos los archivos de audio, extrayendo coeficientes que representen la forma de onda. Para realizar la parametrización, se realizará un enventanado de la señal, y por cada ventana se extraerá un vector de coeficientes. Para ello usaremos el script make_mfcc.sh, este script convierte los archivos en series de coeficientes MFCC siguiendo los segmentos obtenidos usando swbd1_data_prep.sh.

Los *Mel-frequency cepstral coefficients* (MFCCs) son coeficientes que representan una señal de audio, en nuestro caso la voz. Los MFCCs se obtienen mediante un banco de filtros triangulares basados en la escala Mel, una escala perceptual sobre la frecuencia percibida. Kaldi obtiene los MFCCs de la siguiente manera: Primero realizamos un preénfasis y anulación de la señal continua, posteriormente enventanamos la señal con ventanas deslizantes. Enventanamos con una ventana "Povey", parecida a una ventana Hamming pero que llega al valor nulo en los extremos de la ventana. Posteriormente obtenemos la energía de la señal y realizamos la DFT, calculamos la potencia del espectro para poder calcular la energía en 23 de los filtros triangulares de la escala Mel, calculamos el logaritmo de estas energías para pasar al cepstrum de la señal y realizamos la DCT, quedándonos con 13 coeficientes para obtener los vectores deseados [13].

Finalmente, utilizaremos el script subset_data_dir.sh para separar la base de datos en *sets* útiles para los experimentos. Las primeras 4000 frases se usarán como conjunto de evaluación tanto como para los entrenamientos de Kaldi como los realizados en Python usando Theano. Creamos un conjunto de 10000 locuciones cortas para entrenar el sistema basado en fonemas, ya que las locuciones cortas son más fáciles de alinear, lo que ayuda agilizar este tipo de entrenamientos. Por otra parte tendremos un conjunto de 30000 locuciones, que usaremos en los entrenamientos delta+delta-delta. Los últimos dos conjuntos de entrenamiento constan de: un conjunto de 100000 locuciones y otro que consta de la base de datos entera, sin el conjunto de evaluación.

4.3. Sistema de entrenamiento para Kaldi

Al realizar los entrenamientos en Kaldi seguiremos una sucesión de experimentos en la que los resultados de un experimento y sus alineamientos serán utilizados en posteriores experimentos. El ciclo prototipo de datos se muestra en la figura 4.1.

Empezaremos con un entrenamiento usando fonemas y para ello utilizaremos el conjunto de entrenamiento de 10000 locuciones cortas, por la razón explicada en el apartado anterior. Del resultado de este entrenamiento obtendremos sus alineamientos con el conjunto de 30000

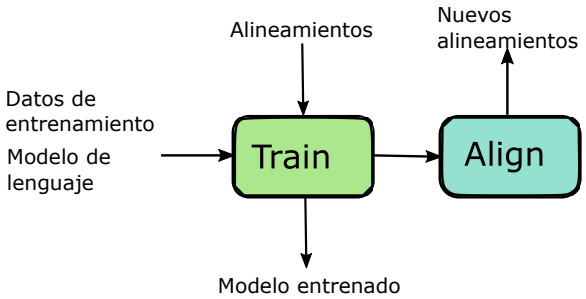

Figura 4.1: Ciclo prototipo

locuciones.

Continuaremos los entrenamientos usando delta+delta-delta sobre el conjunto de 30000 locuciones mediante las alineamientos de los fonemas anteriormente obtenidas. Para ello hacemos uso de train_deltas.sh y obtenemos como resultado nuestro primer entrenamiento de trifonemas, tri1. De nuevo, alineamos el conjunto de 30000 locuciones con el experimento anterior para obtener el directorio tri1_ali, que incluye dichos alineamientos.

A partir de estos alineamientos realizamos un nuevo entrenamiento usando train_deltas.sh y el conjunto de 30000 locuciones. A este entrenamiento lo llamaremos tri2 y sobre su resultado realizaremos un nuevo alineamiento, esta vez con el conjunto de 100000 locuciones.

Sobre el resultado anterior realizaremos un entrenamiento de análisis discriminante lineal y transformación lineal de máxima verosimilitud, LDA+MLLT. Para ello usaremos los alineamientos obtenidos de tri2 con las 100000 locuciones, el resultado obtenido se alineará con el conjunto de 10000 locuciones pero esta vez usaremos el script align_fmllr.sh en vez del script align_si.sh. El script align_fmllr.sh usa un método de regresión lineal llamado *regresión lineal por espacio de características de máxima verosimilitud*. La primera parte de ambos scripts es idéntica, pero align_fmllr.sh realiza dos iteraciones de estimación fMLLR.

Sobre estos alineamientos fMLLR (featured Maximum Likelihood Linear Regression) realizamos un entrenamiento adaptado al locutor, SAT (Speaker Adapted Training). A este entrenamiento se le llamará tri4a. Por otra parte se realizará el proceso de alineamiento anterior con toda la base de datos y se entrenará con el mismo método para dar lugar al experimento tri4b. A partir de este punto se realizarán los alineamientos y entrenamientos en paralelo, tri4a usará el conjunto de 100000 locuciones mientras que tri4b usará el conjunto formado por toda la base de datos.

Siguiendo la cadena, de los dos resultados anteriores obtenemos sus alineamientos, de nuevo usando align_fmllr.sh. Para los próximos experimentos será necesario obtener los látices de los experimentos mediante el script make_denlats.sh. Obtener los látices nos prepara para los experimentos que usan máxima información mutua y que cuyos resultados obtenemos mediante los scripts train_mmi.sh y train_mmi_fmmi.sh.

Como diferencia con respecto a los anteriores entrenamientos, tanto el entrenamiento MMI como el fMMI (featured Maximum Mutual Information) realizan varias iteraciones, intentando ajustarse a los datos de entrenamiento. Finalmente entrenaremos una red neuronal profunda usando la *receta* de Daniel Povey integrada en Kaldi como nnet2. En la figura 4.2 podemos observar el flujo de información entre los experimentos, indicado que conjunto de datos de entrenamiento se usan en cada parte del sistema, así como los alineamientos que se envían a los sistemas de entrenamiento de las redes neuronales profundas.

4.4. Evaluación de los resultados de Kaldi

Para validar los alineamientos y resultados obtenidos, Kaldi incorpora un script llamado decode.sh, este script recibe el grafo resultante de un experimento y un conjunto de datos que deseamos validar, en nuestro caso el conjunto de 4000 locuciones generados al principio del script principal. Existen dos versiones del script *decode* distintas, las cuales también usaremos en este Trabajo Fin de Grado, decode_fmmi.sh y decode_nnet.sh, para obtener las transcripciones de los entrenamientos fMMI y de la red neuronal respectivamente. Los scripts *decode* nos dan los resultados en WER (Word Error Rate).

```
steps/decode.sh [options] <graph-dir> <data-dir> <decode-dir>
```

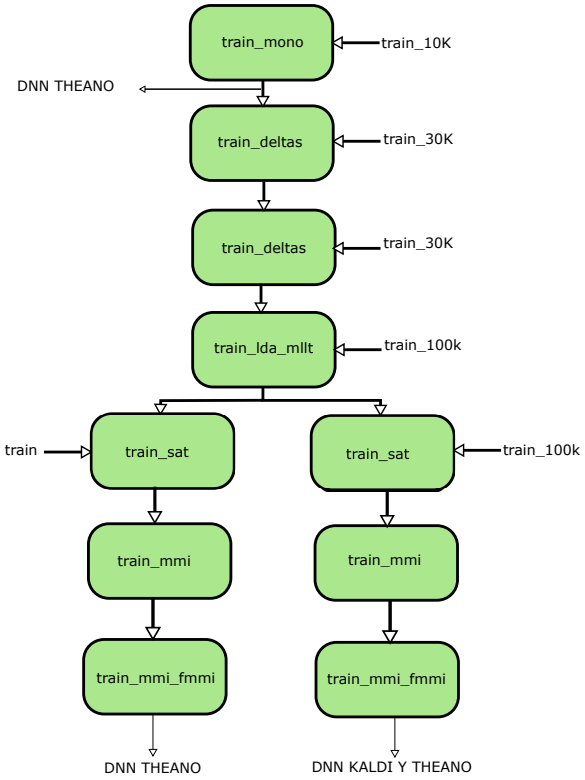

Figura 4.2: Flujo de información

4.5. Sistema de entrenamiento para Theano

El entrenamiento de las redes neuronales realizadas mediante el uso de las librerias de Theano, se ha generado mediante la modificación de un grupo de scripts, aportados por la tutora de este Trabajo Fin de Grado, Alicia Lozano Díez.

El código del entrenamiento se distribuye principalmente entre dos archivos: train_DNN.py y NN_functions.py. El script principal,train_DNN.py, ejecuta las funciones definidas en NN_functions.py para realizar los entrenamientos.

Para entrenar las redes neuronales mediante estos scripts de Python, necesitamos información obtenida al realizar los entrenamientos usando Kaldi. Algunos datos necesarios, que no se generaron en el script principal de Kaldi, se obtienen mediante otras herramientas de Kaldi. Los datos necesarios para entrenar las redes son: las etiquetas de entrenamiento y una lista con los segmentos del conjunto de entrenamiento, para obtener estos datos hemos creado un grupo de scripts en bash y uno en MatLab©, además de usar un ejecutable de Kaldi, show-transitions. Mediante el uso show-transitions obtenemos el *mapeo* de un modelo, este *mapeo* relaciona los identificadores de las transiciones de las unidades fonéticas con sus respectivos identificadores de fonema (pdfs). Usando el archivo generado, los alineamientos y un script en bash llamado tranID2pdfID.sh, generaremos un archivo que en vez de tener los identificadores de las transiciones, tendrá los pdfs de los segmentos de audio que aparecen en los archivos de alineamiento, formando así las etiquetas de entrenamiento de la red.

El script de entrenamiento, train_DNN.py, recibirá por tanto las etiquetas y la lista de segmentos. Al realizar el entrenamiento este será registrado en un archivo, en el que se describe cada paso y los resultados parciales de entrenamiento de error y precisión a nivel fonema enventanado. Por otra parte, el script guarda la matriz de pesos, la tasa de entrenamiento y la reducción de esta ultima para cada iteración, lo que ayuda a recuperar el entrenamiento en caso de fallo.

4.6. Evaluación de los resultados de Theano

Para la evaluación de los entrenamientos en Theano, usaremos un script the Python llamado, validate_DNN.py. En este script cargaremos los pesos de cada iteración y validaremos el conjunto de evaluación, de nuevo usando las etiquetas y lista de segmentos del conjunto de evaluación (4000 locuciones), obtenidos de la misma manera que en la preparación para los entrenamientos. El resultado se obtendrá en error y precisión a nivel fonema enventanado.

5

Experimentos y Resultados

5.1. Sistema basado en fonemas usando alineamiento de MFCC

El primer experimento que realizaremos será un entrenamiento alineando los MFCCs del conjunto de entrenamiento. Usaremos sus etiquetas de fonemas para generar un GMM que sea capaz de representar los fonemas como distintas clases.

Primero, para normalizar el canal e intentar anular las variaciones que pueden causar problemas a la hora de hallar el GMM, calcularemos dos medidas estadísticas: la media y varianza cepstral normalizadas. Ambas son calculadas a partir de los MFCC de las locuciones de entrenamiento del conjunto de 10000 locuciones cortas.

Posteriormente, generaremos un GMM que intente representar los fonemas de las locuciones de entrenamiento realizando sucesivos alineamientos entre las etiquetas del conjunto y sus MFCCs, adaptados con los estadísticos provenientes del algoritmo de *cepstral mean and variance normalization* (CMVN). Finalmente el resultado obtenido es el que aparece en el cuadro 5.1.

Experimento	WER
Fonemas med. MFCC	78.25 %

Cuadro 5.1: Resultados entrenamiento para fonemas usando sólo los MFCC

Un resultado de casi el 80 % de tasa de error es muy elevado. Para disminuir la tasa de error, los experimentos de los próximos apartados se basan en sistemas más complejos.

5.2. Sistema basado en trifonemas usando información de deltas

Para este experimento, incluiremos información temporal de los coeficientes estáticos MFCC, es decir, su primera y segunda derivada (velocidad y aceleración). Teniendo en cuenta que trabajamos en un espacio discreto, la función derivada no esta definida como tal. Para solventar este problema, usaremos la función diferencia en lugar de la derivada. Debido al símbolo de la función diferencia, a este tipo de entrenamientos se les denomina delta y delta-delta.

En Kaldi usaremos add-deltas para calcular ambos parámetros, primero calculando el parámetro delta con una ventana deslizante normalizada, y sobre los parámetros obtenidos se realiza de nuevo la misma operación para obtener los parámetros delta-delta [14]. La fórmula usada para obtener el coeficiente delta de un instante i es la ecuación 5.1.

$$y_i = \frac{(x_{i-2} \times -2) - x_{i-1} + x_{i+1} + (x_{i+2} \times 2)}{10} \tag{5.1}$$

Donde x_i es el coeficiente en el instante i e y_i es el parámetro delta en el mismo instante.

Utilizando esta nueva parametrización, generaremos un GMM y un árbol de decisión para realizar el etiquetado en trifonemas del conjunto de 30000 locuciones (3 veces más locuciones que para entrenar fonemas).

Posteriormente y con los alineamientos generados a partir del modelo delta-delta anterior, realizaremos un nuevo entrenamiento delta-delta del conjunto de 30000 locuciones.

Como podemos observar en el cuadro 5.2, el resultado del entrenamiento delta-delta reduce en casi 35 % el WER con respecto al entrenamiento de alineamiento de MFCCs usando fonemas. Por otra parte, realizar el mismo entrenamiento sobre el modelo delta-delta proveniente de los alineamientos MFCCS usando fonemas, reduce un 0.7 % el WER.

Experimento	WER
Δ sobre fonemas	44.99 %
Δ sobre Δ	44.29 %

Cuadro 5.2: Resultados entrenamiento mediante delta y delta-delta

5.3. Sistema basado en trifonemas usando LDA y MLLT

En el siguiente experimento utilizaremos Linear Discriminant Analysis (LDA) y Maximum Likelihood Linear Tranformation (MLLT) para, a partir de los alineamientos del experimento anterior, obtener un GMM y árbol de decisión para etiquetar los trifonemas. Como comentamos en el estado del arte, mediante LDA [15] se busca una transformación lineal que sea capaz de discriminar entre las distintas clases. Para realizar este proceso, se busca una matriz de proyección que permite ver las distintas clases más separadas, en otras coordenadas.

Por otra parte MLLT busca maximizar la covarianza entre los vectores de coeficientes, lo que nos permite mantener la intravariabilidad, es decir, variabilidad para un mismo locutor. El resultado de este entrenamiento queda representado en el cuadro 5.3.

Este resultado presenta una mejora de casi el 8 % sobre los anteriores experimentos. Podemos ver como mejora la función decorrelante de LDA este tipo de sistemas, por ello, se utilizarán funciones de este tipo en la preparación de los datos de las DNNs.

Experimento	WER
LDA+MLLT	36.19 %

Cuadro 5.3: Resultados entrenamiento mediante LDA+MLLT

5.4. Sistema basado en trifonemas usando SAT

El entrenamiento de adaptación al locutor consiste en buscar las similitudes y diferencias entre los locutores [16], intentando así separar la variabilidad de los locutores de la variabilidad fonética. Los resultados de este entrenamiento aparecen en el cuadro 5.4, tanto como para el entrenamiento con el conjunto de 100000 locuciones como el de la base de datos en su totalidad.

Experimento	WER
SAT sobre 100k	31.42 %
SAT sobre train	29.41 %

Cuadro 5.4: Resultados entrenamiento mediante LDA+MLLT+SAT usando el conjunto de 100000 locuciones y el conjunto de entrenamiento en su totalidad

Con estos resultados podemos ver como afecta el tamaño del conjunto de entrenamiento a los resultados, realizando el mismo entrenamiento, sobre un conjunto más grande logramos reducir la tasa de error, en este caso en un 2 % con aproximadamente el doble de locuciones en el conjunto de entrenamiento entero, sobre el de 100000 locuciones.

5.5. Sistema basado en trifonemas usando MMI

En este experimento buscaremos, mediante el script train_mmi.sh, maximizar la información mutua teniendo en cuenta un parámetro *boost*, de valor 0.5 durante estos experimentos. Este entrenamiento nos permite aumentar la verosimilitud de los caminos que contienen más errores [17]. El criterio utilizado para este experimento es la información mutua entre las probabilidades de observación y las secuencias de palabras, para ello usamos la ecuación 5.2.

$$F_{MMI} = \sum \log \frac{p(\mathbf{O}|S_u)^{\kappa} P(W_u)}{\sum_W p(\mathbf{O}_u|S)^{\kappa} P(W) e^{-bA(W,W_u)}} \tag{5.2}$$

Donde $\mathbf{O}_u$ es la secuencia de todas las observaciones, W_u es la secuencias de palabras para la locución u, S_u es la secuencia de estados correspondiente a W_u, k es el factor de escalado acústico y b el factor de *boosting*.

Los resultados de estos entrenamientos aparecen reflejados en los cuadros 5.5 y 5.6, donde el WER es acompañado del número de la iteración donde fue conseguido.

En estos resultados podemos observar como el aumentar el número de iteraciones tiende a disminuir la tasa de error, pero existe un punto en el que el aumento de iteraciones empeora el resultado. En este caso una cuarta iteración nos aleja de la tasa de error mínima para el experimento que usa como entrenamiento el conjunto de datos en su totalidad. Si ahora representamos en una gráfica todos los resultados juntos obtenemos la siguiente representación de la figura 5.1.

Iteración	WER
1	29.41 %
2	28.58 %
3	28.44 %
4	28.21 %

Cuadro 5.5: Resultados del entrenamiento MMI para el conjunto de 100000 locuciones

Iteración	WER
1	27.17 %
2	26.52 %
3	26.40 %
4	26.49 %

Cuadro 5.6: Resultados del entrenamiento MMI para el conjunto de entrenamiento en su totalidad

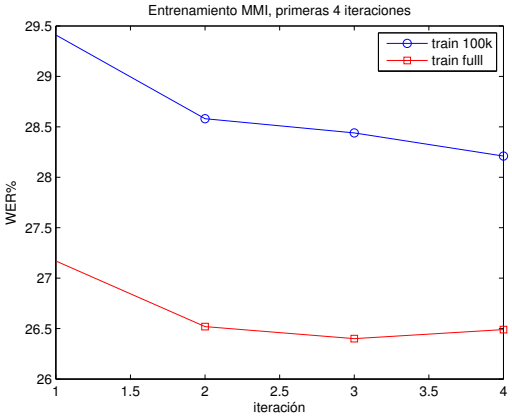

Figura 5.1: Resultados MMI

5.6. Sistema basado en trifonemas usando fMMI

El último experimento realizado antes de los entrenamientos de redes neuronales profundas consiste en una versión modificada del anterior, fMMI es la realización de MMI introduciendo componentes del espacio del modelo y el espacio de características. Mediante esta manipulación logramos que las características más parecidas definan clases más compactas.

En el cuadro 5.7 y 5.8 podemos observar los resultados de este entrenamiento usando el conjunto de 100000 locuciones y la base de datos en su totalidad como conjuntos de entrenamiento.

Si representamos ambas series de resultados gráficamente (Figura 5.2), obtenemos una representación en la que podemos ver la diferencia causada por un entrenamiento con más datos. También podemos observar como aparece un punto de inflexión en el WER alrededor de la sexta iteración.

Iteración	WER
4	29.84 %
5	28.58 %
6	27.86 %
7	28.06 %
8	27.88 %

Cuadro 5.7: Resultados del entrenamiento fMMI para el conjunto de entrenamiento de 100000 locuciones

Iteración	WER
4	28.07 %
5	26.53 %
6	25.97 %
7	25.99 %
8	26.28 %

Cuadro 5.8: Resultados del entrenamiento fMMI para el conjunto de entrenamiento en su totalidad

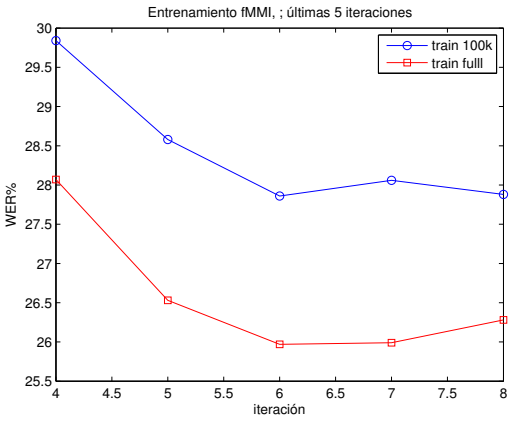

Figura 5.2: Resultados fMMI

5.7. Comparación de los resultados previos a las DNNs

A primera vista podemos ver como a medida que los experimentos aumentan su complejidad la tasa de error por palabra disminuye. En la figura 5.3 buscamos representar esta reducción de la tasa de error sin tener en cuenta si trabajamos con fonemas o trifonemas.

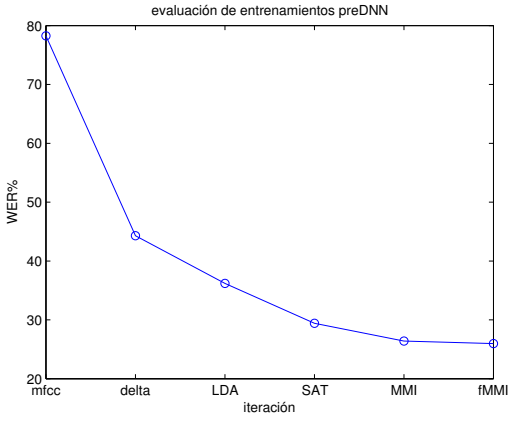

Figura 5.3: Resultados sin redes neuronales

También podemos ver como el entrenamiento basado en fonemas da peores resultados que el entrenamiento en trifonemas más básico. Incluso teniendo en cuenta que el entrenamiento más básico para trifonemas es mas complejo que el entrenamiento basado en fonemas realizado. la diferencia tan grande en la tasa de error apoya la hipótesis inicial de que el entrenamiento por trifonemas ofrece mejores resultados que el entrenamiento basado en fonemas.

Por otra parte, podemos ver como dentro los entrenamientos en los que se realizan varias iteraciones, al aumentar el número de estas mejora la tasa de error, pero existe un punto de inflexión en los resultados. Al aumentar el número de iteraciones podemos alejarnos del resultado óptimo, esto se debe a que se empieza a sufrir de sobreentrenamiento, es decir, nos ajustamos demasiado a los datos de entrenamiento y perdemos la generalidad necesaria para decodificar correctamente datos distintos a los datos con los que hemos entrenado.

5.8. Sistema basado en la DNN de Kaldi

Para la realización del entrenamiento de la red neuronal mediante las herramientas proporcionadas por Kaldi, usaremos la *receta* propuesta por Daniel Povey [18]. Esta red recibe como entrada el modelo generado por las características MFCC+LDA+MLLT+fMMI provenientes del experimento anterior. Estas característica de 40 dimensiones, son entregadas a la red en ventanas de 5 *frames*, que son multiplicadas por una transformación decorrelante basada en LDA.

Esta red funciona usando como función de activación la tangente hiperbólica. La tangente hiperbólica se define formalmente mediante la fórmula 5.3.

$$tanh\ x = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{5.3}$$

Esta fórmula nos define la gráfica de la figura 5.4.

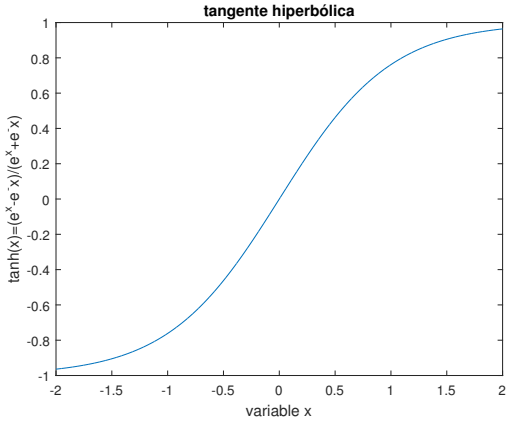

Figura 5.4: Tangente hiperbólica entre -2 y 2

Como podemos observar la tangente hiperbólica y la función sigmoide se parecen bastante. La mayor diferencia entre ambas se encuentra en la zona de máxima pendiente, donde la tangente hiperbólica tiene una derivada de mayor valor que la función sigmoide.

El entrenamiento constara en minimizar el coste usando descenso por gradiente estocástico. La fórmula 5.4 indica como se actualizan los parámetros por cada iteración de entrenamiento.

$$\theta = \theta - \alpha \nabla_\theta (\theta; x^{(i)}, y^{(i)}) \tag{5.4}$$

Donde $x^{(i)}$ e $y^{(i)}$ es una pareja de características provenientes del conjunto de entrenamiento y la función θ la función de coste.

Para poder realizar correctamente el algoritmo de *Stochastic Gradient Descent* (SGD), aleatorizamos las entradas, a nivel ventana, una vez. Este proceso ayuda a que el algoritmo SGD

no converja a niveles erróneos, este problema suele surgir por usar entradas del mismo signo de forma constante.

Inicialmente la red neuronal se genera con una sola capa, la cual se expandirá hasta las 4 capas (de 1024 neuronas cada una) a medida que avanza el entrenamiento. El ejecutable nnet-am-init creara el modelo inicial, con una configuración tal que:

```
SpliceComponent input-dim=40 left-context=4 right-context=4
FixedAffineComponent matrix=exp/nnet5a/lda.mat
AffineComponentPreconditioned input-dim=360 output-dim=1024 alpha=4.0 max-
    change=10.0 learning-rate=0.01 param-stddev=0.03125 bias-stddev=0.5
TanhComponent dim=1024
AffineComponentPreconditioned input-dim=1024 output-dim=4258 alpha=4.0 max-
    change=10.0 learning-rate=0.01 param-stddev=0 bias-stddev=0
SoftmaxComponent dim=4258
```

Donde podemos observar un elemento llamado AffineComponent, el cual representa la matriz de pesos y *bias*. El *bias* es un valor que le permite a las neuronas desplazar la función de activación en el eje x. También podemos encontrar la cadena TanhComponent, que indica el uso de una tangente hiperbólica como función no lineal de activación para la red, como se ha explicado con anterioridad.

Siguiendo la ejecución del script aparece el ejecutable nnet-train-transitions, el cual prepara los HMMs necesarios para la validación, así como las probabilidades *a priori* de las clases.

El entrenamiento de la red consiste en un búcle de 20 épocas, 15 en las que reducimos la tasa de entrenamiento y las 5 últimas en las que la mantenemos constante. Cada época consta de un número de iteraciones que viene definida por el número de datos, cuantos más datos mayor es el número de iteraciones. Al iniciar cada iteración lo primero que se realiza es el cálculo de una serie de datos de diagnóstico, por ejemplo, para la primera y última iteración de la red neuronal entrenada son los siguientes:

```
exp/nnet5a/log/compute_prob_train.0.log:
LOG (nnet-compute-prob:main():nnet-compute-prob.cc:91) Saw 4000 examples,
average probability is -8.35648 and accuracy is 0.066  with total weight
    4000
exp/nnet5a/log/compute_prob_train.final.log:
LOG (nnet-compute-prob:main():nnet-compute-prob.cc:91) Saw 4000 examples,
average probability is -1.59958 and accuracy is 0.572 with total weight
    4000
```

Como se puede observar estos datos de diagnóstico nos dan un indicativo de que la red se ha entrenado de forma correcta. El *log* indica que el *negative log likelihood* de la primera iteración es mayor que la de la última, y la precisión de la última es mayor que para la primera. Existe otro archivo *log*, progress.log, que nos permite detectar las etapas internas de entrenamiento por iteración:

```
At position 0.5, objf per frame is
-1.65148
Total objf change per component is
[ 0.000824921 -0.00153218 -5.05503e-05 -0.0020028 8.86936e-05 ]
Parameter differences per layer are
[ 0.94477 4.72312 4.90913 4.17734 1.94486 ]
Relative parameter differences per layer are
[ 0.00370865 0.0775327 0.109199 0.100262 0.00329796 ]
```

Usando este *log*, podemos ver como los parámetros internos de la red varían por cada iteración. Por otra parte hacemos un *log* de los pasos de entrenamiento por cada iteración e hilo:

```
nnet-train-parallel --num-threads=16 --minibatch-size=128 --srand=779 exp/
    nnet5a/779.mdl ark:- exp/nnet5a/780.1.mdl
nnet-shuffle-egs --buffer-size=5000 --srand=779 ark:exp/nnet5a/egs/egs
    .1.38.ark ark:-
LOG (nnet-shuffle-egs:main():nnet-shuffle-egs.cc:102) Shuffled order of
    200587 neural-network training examples using a buffer (partial
    randomization)
LOG (nnet-train-parallel:DoBackpropParallel():nnet-update-parallel.cc:194)
    Did backprop on 200587 examples, average log-prob per frame is -1.7228
LOG (nnet-train-parallel:DoBackpropParallel():nnet-update-parallel.cc:196)
    [this line is to be parsed by a script:] log-prob-per-frame=-1.7228
LOG (nnet-train-parallel:main():nnet-train-parallel.cc:102) Finished
    training, processed 200587 training examples (weighted). Wrote model to
     exp/nnet5a/780.1.mdl
# Accounting: time=1126 threads=1
# Ended (code 0) at Wed Feb 24 03:42:59 CET 2016, elapsed time 1126 seconds
```

En este *log* podemos observar como por cada modelo parcial, usamos entradas aleatorizadas y realizamos las respectivas propagaciones. También podemos ver el tiempo de cómputo de estas operaciones, en este caso de 1126 segundos.

Al finalizar las iteraciones de entrenamiento, todos los modelos parciales generados en cada iteración son reunidos en un único modelo llamado final.mdl

Haremos la evaluación de la red usando el conjunto con el que hemos validado el resto de experimentos (conjunto de 4000 locuciones). El resultado obtenido aparece en el cuadro 5.8.

Experimento	WER
DNN Kaldi tri4a-fMMI	24.35 %

Cuadro 5.9: Resultado de la DNN entrenada usando Kaldi

Esta red neuronal ha sido entrenada usando un servidor CPU, de 12 núcleos y 48GB de memoria RAM, compartido entre varios usuarios. Para evitar el colapso de los servicios ofrecidos por el servidor, el entrenamiento se limitó a un total de 5 hilos en paralelo. El tiempo de entrenamiento de la red neuronal fue de aproximadamente 230 horas.

5.9. Sistema basado en DNNs de Theano

Para empezar a realizar los experimentos con las redes de Theano, primero preparamos los datos provenientes de los experimentos de Kaldi. Los datos son transformados en coeficientes HTK (Hidden Markov Model ToolKit). Los coeficientes HTK incluyen la información delta y delta-delta, lo que permite utilizar estos coeficientes en los entrenamientos de Theano.

Lo primero que realizamos es calcular dos estadísticos sobre los datos de entrenamiento, la media y la varianza. Para agilizar el proceso realizamos estos cálculos sobre un conjunto de datos más pequeños que el original, recogiendo una serie de datos cada 1000. Podemos asumir que esta lista reducida de datos representa, estadísticamente, los datos en su totalidad.

Posteriormente mediante la concatenación de los métodos init_params y create_theano_nn de la clase NN_functions generamos el esqueleto de la red neuronal. El método init_params es el encargado de recibir la media y la varianza, así como el número de clases y de capas de la red neuronal, para genera el *diccionario* con el cual create_theano_nn es capaz de generar los parámetros que definen la red neuronal.

El valor de coste de la red estará definido por la entropía cruzada, para ello usaremos el método categorical_crossentropy, de la clase nnet, de las librerías de Theano. El método categorical_crossentropy calcula la entropía cruzada entre dos distribuciones de probabilidad. En nuestro caso serán los datos de entrenamiento como distribución de entrada y los alineamientos como distribución objetivo. La entropía cruzada se calcula como indica la fórmula 5.5.

$$H(p,q) = -\sum p(x)\log(q(x)) \tag{5.5}$$

Donde p es la distribución objetivo y q es la distribución a comparar. Posteriormente aproximaremos los costes mediante el método grad de la clase gradient.

Para los entrenamientos usando Theano hemos decidido usar 12 épocas, ya que empíricamente el resultado convergía a partir de la décima iteración.

Para que fuera más fácil comparar entre las redes de Kaldi y Theano hemos realizado entrenamientos sobre los mismo datos que entrenamos la red de Kaldi. Además hemos realizado dos entrenamientos adicionales, uno usando los alineamientos del entrenamiento usando fonemas, y otro con los alineamientos del entrenamiento fMMI, que usaba el conjunto de entrenamiento en su totalidad. Realizaremos la validación con el mismo conjunto que para el resto de experimentos. El resultado ha sido representado en el cuadro 5.10.

Experimento	Precisión por ventana	Duración	Número de clases
Tri4a fMMI	43 %	3h24m	4258
Tri4b fMMI	45 %	13h59m	8928
Fonemas	48 %	17h36m	146

Cuadro 5.10: Resultados de los entrenamientos de redes neuronales usando las librerías de Theano

Observando los resultados vemos que la mayor precisión a nivel de enventanado fonético, pertenece a la red entrenada mediante los alineamientos de fonemas. A pesar de que este dato parece el mejor, se ha de tener primero en cuenta el número de clases de salida, 146, que le da relatividad al valor de precisión. Se entiende, que para un mayor número de clases, una precisión menor puede ser un mejor resultado. Por tanto, el mejor resultado realmente es el obtenido en la red neuronal entrenada con los alineamientos fMMI y con todo el conjunto de entrenamiento como entrada. Aseguramos este hecho ya que, para casi el doble de clases que el entrenamiento de la red entrenada usando fMMI con el conjunto de 100000 locuciones, obtenemos un resultado un 2 % mayor.

El entrenamiento de las redes neuronales mediante las librerías de Theano se ha realizado usando GPU, excepto el entrenamiento usando los alineamientos de fonemas, que se ha realizado en CPU. Sobre los experimentos basados en fMMi, se han realizado un número pequeño de iteraciones sobre CPU, por lo que se puede realizar una comparación sobre el tiempo de cómputo entre GPU y CPU para estos algoritmos. Estos datos han sido reflejados en el cuadro 5.11.

Arquitectura	Duración de 1 época
CPU	2356m
GPU	25m

Cuadro 5.11: Comparación GPU vs. CPU para el entrenamiento de la red basada en los alineamientos de tri4a fMMI

El resultado es el esperado, la computación por GPU, por las razones establecidas en el estado del arte de este Trabajo Fin de Grado, es mucho más rápida que en CPU, en este caso casi 100 veces más rápida. El entrenamiento de la red neuronal basada en los alineamientos de

Tri4a fMMI tardó 3 horas y 25 minutos usando GPU, en CPU podría haber tardado casi 20 días. Uno de los principales problemas a la hora de desarrollar este Trabajo Fin de Grado han sido las constricciones temporales para obtener resultados, cada resultado de una DNN en CPU puede tardar en el orden de dos semana en conseguirse si se usa toda la potencia posible de los núcleos CPU en sólo esta tarea.

5.10. Comparación de todos los resultados

Como podemos observar nuestra hipótesis es corroborada. Las redes neuronales profundas son capaces de obtener mejores resultados que los métodos clásicos de entrenamiento, si hacemos una comparación de todos los datos obtenidos en los experimentos de Kaldi, obtenemos la figura 5.5.

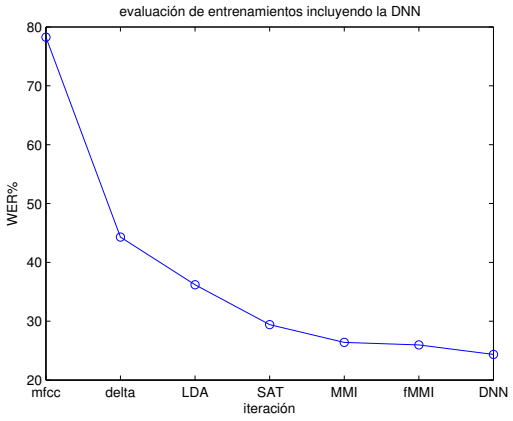

Figura 5.5: Resultados de los experimentos de Kaldi incluyendo la DNN

Dentro de las redes neuronales profundas vemos como un mayor número de datos de entrenamiento mejora los resultados, comparando los resultados de las redes neuronales de Theano en la figura 5.6, veremos como incluso para un mayor número de clases de salida, esto es, trifonemas o fonemas observados en el entrenamiento, la precisión por ventana es mejor para el mismo número de épocas de entrenamiento.

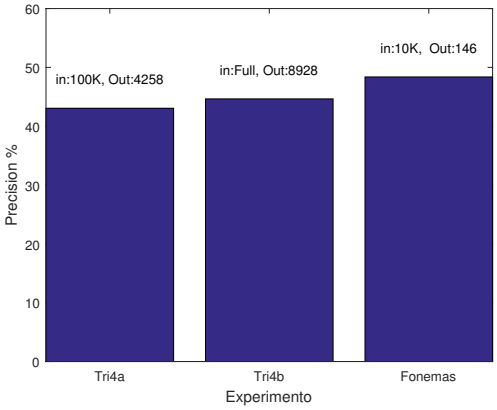

Figura 5.6: Resultados de las redes de Theano acompañadas de número de datos de entrada y clases de salida

También podemos analizar los resultados por épocas para un entrenamiento de las redes de Theano, por ejemplo para el entrenamiento usando todo el conjunto de entrenamiento, podemos observar (ver figura 5.7), como este converge al resultado obtenido a medida que avanzan las épocas.

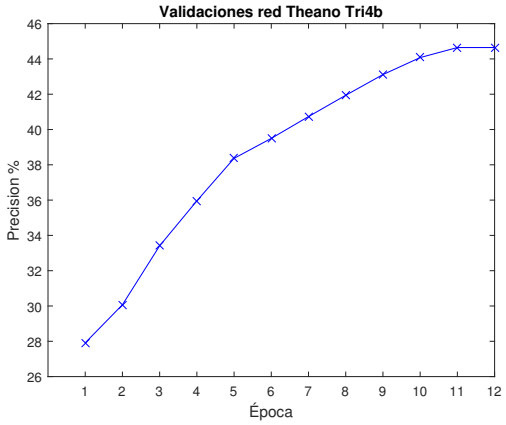

Figura 5.7: Evaluaciones por época de la red de Theano entrenada con los alineamientos de fMMI

Si ahora comparamos las decodificaciones del sistema fonema y el de la red neuronal entrenada en Kaldi, con la transcripción original de una locución, podemos ver la mejora a nivel de inteligibilidad:

Monofonema: right i think there is there um is there a little bit of fort worth it
DNN: right right is there is there um as there is like a coda dressed where you work to have
Transcipcion: right right is there is there um an– is there a like a code of dress where you work do they ask

Si intentamos leer el texto decodificado, nos damos cuenta de que para el entrenamiento usando fonemas, casi no somos capaces de entenderlo. Si pasamos al resultado de la DNN entrenada mediante trifonemas, observamos como a pesar de que el texto decodificado no es igual a la transcripción original, este es más inteligible que el anterior.

6

Conclusiones y trabajo futuro

La primera conclusión a la que llegamos con los experimentos realizados es la confirmación de las hipótesis expuestas en la introducción. Hemos reflejado con los resultados obtenidos, cómo las redes neuronales profundas son capaces de mejorar los sistemas de etiquetado de voz tradicionales, además, ha quedado también demostrado que para etiquetar segmentos de audio es mejor utilizar sistemas basados en trifonemas, ya que los sistemas basados en fonemas presentan una mayor probabilidad de error.

También hemos observado como el tipo de arquitectura de cálculo usada en el entrenamiento de las redes neuronales, afecta al tiempo necesario para dichos entrenamientos. Con una diferencia temporal de hasta 100 veces menor usando GPU, hemos demostrado que para el manejo de entrenamientos con volúmenes de información altos es necesario utilizar este tipo de arquitectura.

Cabe destacar el uso de las herramientas empleadas, Kaldi y Theano, mediante las cuales el proceso del tratamiento de voz y entrenamiento de las redes ha sido mucho más organizado y efectivo. Kaldi nos ha permitido obtener los alineamientos necesarios para los experimentos realizados. Las librerías de Theano han sido diseñadas con *Deep Learning* como uno de los objetivos principales: este hecho se traduce en clases con métodos que se pueden aplicar directamente a los algoritmos de entrenamiento de DNNs. Estos métodos agilizan el trabajo con funciones de coste y producto de matrices.

Mi labor en este Trabajo Fin de Grado se puede dividir en dos partes. La primera ha sido, en los experimentos de Kaldi, adaptar los ya existentes scripts del ejemplo de switchboard *s5b*, al entorno de trabajo de los laboratorios de ATVS. Esto incluye la modificación de los *paths*, los archivos de evaluación y el balance de número de tareas paralelas para no sobrecargar el servidor. La segunda parte ha sido, modificar los scripts otorgados por Alicia Lozano, los cuales creaban una red tipo *bottleneck*, para crear una red neuronal simétrica. Esta modificación exigía eliminar la extracción intermedia de características que la red *bottleneck* realizaba.

Como trabajo futuro dentro del ámbito de este Trabajo Fin de Grado, sería importante buscar una implementación conjunta entre las herramientas de Kaldi y Theano, es decir, convertir la obtención de los alineamientos, usando Kaldi, y entrenamiento, usando Theano, en un solo script ejecutable.

Por otra parte, convertir los resultados de las redes entrenadas usando Theano, en valores WER, permitiría comparar mejor entre los resultados de las redes de Kaldi y las de Theano.

Además, obtener los resultados en WER, permite entender mejor la fiabilidad de la red.

Finalmente, ajustar el número de capas y neuronas mediante la repetición de experimentos, nos permitiría encontrar de forma empírica una red que nos ofreciera mejores resultados. Posteriormente, experimentar con otros tipos de redes, como pueden ser redes convolucionales y redes recurrentes [19], para así comprobar si estas redes mejoran los resultados obtenidos en este Trabajo Fin de Grado.

Las redes convolucionales intenta explotar las características conjuntas de los datos de entrada, por lo que es preferible que sus datos de entrada tenga más de una dimensión. Estas redes se organizan en una serie de etapas, las primeras etapas se componen de capas convolucionales y capas acumuladoras. Estas capas se comunican entre ellas compartiendo características.

Si exploramos las líneas futuras de investigación sobre las DNNs, debemos hablar de las redes convolucionales y recurrentes.

Las redes recurrentes, son redes en las que una neurona se puede comunicar consigo misma. Para evitar caer el en problema de que la estimación de pesos entre en un búcle que neutralize o le de un peso excesivo a una neurona, se usan *memorias*. Las redes tipo *Long Short Term Memory networks* (LSTM nets), son redes recurrentes, en las que las conexiones entre neuronas y con la propia neurona están gobernadas por una puerta lógica que se activa o no dependiendo de los pesos propagados. Estas redes han sido usadas recientemente en conjunto con una red convolucional para el reconocimiento de idioma. Mediante una capa oculta de menor tamaño (capa *bottleneck*) de la red convolucional, se extraen características que serán utilizadas para entrenar una LSTM-RNN (*Long Short Term Memory-Recurrent Neural Network* [20].

Glosario de acrónimos

- **DNN**: Deep Neural Network
- **DTW**: Dynamic Time Warping
- **GMM**: Gaussian Mixture Model
- **HMM**: Hidden Markov Model
- **UBM**: Universal Background Model
- **LDC**: Linguistic Data Consortium
- **LDA**: Linear Discriminant Analysis
- **MFCC**: Mel-Frecuency Cepstral Coefficient
- **HTK**: Hidden markov model ToolKit
- **MMI**: Maximum Mutual Information
- **MLLT**: Maximum Likelihood Linear Transform
- **MLLR**: Maximum Likelihood Linear Regression
- **CPU**: Central Processing Unit
- **GPU**: Graphics Processing Unit

Bibliografía

[1] Thomas D. Rossing et al. Springer handbook of acoustics, 2008.

[2] Yiteng Huang(Eds.) Jacob Benesty, M. Mohan Sondhi. *Springer Handbook of Speech Processing*. 2008.

[3] International Phonetic Association. The international phonetic alphabet (revised to 2015), 2015.

[4] Meinard Müller. *Information retrieval for music and motion*. 2007.

[5] Chunsheng Fang. From dynamic time warping (dtw) to hidden markov model (hmm), 2009-3-19.

[6] Yiteng Huang(Eds.) Jacob Benesty, M. Mohan Sondhi. *Springer Handbook of Speech Processing*. 2008.

[7] The open standard for parallel programming of heterogeneous systems. `https://www.khronos.org/opencl/`. Accessed: 2016-06-14.

[8] Nvidia cudnn gpu accelerated deep learning. `https://developer.nvidia.com/cudnn`. Accessed: 2016-05-20.

[9] History of the kaldi project. `http://kaldi-asr.org/doc/history.html`. Accessed: 2016-05-19.

[10] Yajie Miao. Kaldi+pdnn: Building dnn-based asr systems with kaldi and pdnn. *ARXIV*, 01 2014.

[11] Welcome - theano 0.8.2 documentation. `http://deeplearning.net/software/theano/`. Accessed: 2016-05-19.

[12] Switchboard-1 release 2. `https://catalog.ldc.upenn.edu/docs/LDC97S62/`. Accessed: 2016-06-15.

[13] Kaldi: Feature extraction. `http://www.danielpovey.com/kaldi-docs/feat.html`. Accessed: 2016-05-27.

[14] Feature and model-space transforms in kaldi. `http://kaldi-asr.org/doc2/transform.html#transform_delta`. Accessed: 2016-05-24.

[15] *Regularizing Linear Discriminant Analysis for Speech Recognition*, 2005. in Proc. INTERSPEECH.

[16] Juri Ganitkevitch. Speaker adaptation using maximum likelihood linear regression, 2005.

[17] ISCA. *Sequence-discriminative training of deep neural networks*, August, 2013. in Proc. INTERSPEECH.

[18] Kaldi: Dan's dnn implementation. `http://www.danielpovey.com/kaldi-docs/dnn2.html`. Accessed: 2016-05-25.

[19] Hinton Geoffrey Yann LeCun, Bengio Yoshua. Deep learning. *Nature*, pages 439–442, 2015.

[20] Ruben Zazo, Alicia Lozano-Diez, Javier Gonzalez-Dominguez, Doroteo T. Toledano, and Joaquin Gonzalez-Rodriguez. Language identification in short utterances using long short-term memory (lstm) recurrent neural networks. *PLoS ONE*, 11(1):1–17, 01 2016.

A

Entorno de trabajo

Para la realización de este Trabajo Fin de Grado, se ha trabajado usando 2 ordenadores personales y 1 servidor. Los 3 equipos pertenencen al laboratorio de biométrica de la Universidad Autónoma de Madrid (ATVS).

- atvs-Vostro-260:
  - CPU: Intel® Core™ i5-2400 CPU @ 3.10GHz.
  - GPU: ATI Radeon™ RV370 [Radeon X300/X550/X1050 Series]
  - RAM: 4 GB
  - ROM: 1 TB
- atvsBiometria:
  - CPU: Intel® Core™ i7-6700K CPU @ 4.00GHz.
  - GPU: Nvidia GeForce GTX 970, 4GB GDDR5
  - RAM: 32GB
  - ROM: 2 TB
- s15:
  - CPU: 2x Intel® Xeon® CPU E5649 @ 2.53GHz
  - GPU: VGA compatible controller by ASPEED
  - RAM: 48 GB
  - ROM: 781 GB

Los entrenamientos en CPU se realizaron en el servidor s15. Los entrenamientos en GPU usaron el equipo atvsBiometria, esto se debe a su potente GPU, la cual nos ha permitido reducir el tiempo de entrenamiento comparandolo con los entrenamientos en CPU.

El equipo atvs-Vostro-260 se ha utilizado principalmente para la redacción de código y de la memoria del Trabajo Fin de Grado, además de ser el punto de acceso prioritario a los otros dos equipos. Todos los equipos usan como sistema operativo Ubuntu.

B

Manual del programador

Para poder reproducir estos experimentos se ha de ejecutar el script bash run.sh. Este script consta de una versión modificada del script del mismo nombre del ejemplo switchboard (versión s5b) de Kaldi. Esta versión permite la correcta ejecución en el entorno de trabajo de este Trabajo Fin de Grado, es decir, en los equipos de ATVS mencionados en el Anexo A.

Para ejecutar esta versión de run.sh el script se ha de ejecutar sin argumentos. Si se necesita cambiar algún *path* se debe de editar el script o el archivo de path.sh, dependiendo de el *path* que se necesite cambiar. Si se desea usar otra base de datos para la evaluación de los sistemas generados se deben cambiar las llamadas referentes a train_dev, para hacer referencia a esta nueva base de datos de validación. Si además se desean obtener los resultados en WER para lo experimentos de Kaldi, es necesario tener las transcripciones en l formato definido en el primer subapartado de *Diseño y desarrollo* de este Trabajo Fin de Grado.

Para la ejecución de los scripts de Python es necesario instalar Theano y numPy (junto con otras herramientas de Python). Para ello se recomienda el uso de las siguientes líneas de código bash para su correcta instalación en Ubuntu:

```
sudo apt-get install python-numpy python-scipy python-dev python-pip python
    -nose g++ libblas-dev git
pip install --upgrade --no-deps git+git://github.com/Theano/Theano.git --
    user
```

Ejecutar el script train_DNN.py, permite el entrenamiento de una red neuronal profunda usando los datos que aparecen en los *paths* al inicio del archivo. Los datos han de ser manipulados como se explica en el apartado 4.5 de este Trabajo Fin de Grado, para la correcta interpretación de esto por parte de los algoritmos de entrenamiento. El entrenamiento esta preparado para funcionar en GPU, si se desea cambiar el tipo de arquitectura a usar basta con cambiar el*flag* de device=gpu a device=cpu. Este *flag* forma parte de 'THEANO_FLAGS', variable de entorno que podemos encontrar definida al principio del script train_DNN.py.