Escuela Politécnica Superior de Linares

GRADO EN INGENIERÍA DE TECNOLOGÍAS DE LA TELECOMUNICACIÓN

Universidad de Jaén

Escuela Politécnica Superior de Linares

SISTEMA DE TRANSCRIPCIÓN MUSICAL BASADO EN DEEP LEARNING

Alumno: Francisco Antonio Cartas Martínez

Tutores: Prof. D. Julio José Carabias Orti

Prof. D. Pedro Vera Candeas

Depto.: Ingeniería de Telecomunicación

ÍNDICE DE CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1. INTRODUCCIÓN

Las tecnologías y la ciencia de la información se encuentran a la orden del día en la vida cotidiana, creciendo de manera impresionante, paralelamente al gran avance tecnológico de estas últimas dos décadas principalmente.

1.1 RESUMEN

Para comenzar a detallar en que consiste este trabajo, se expone un breve viaje sobre en qué va a consistir dicho proyecto.

Este proyecto consiste, de forma general, en un sistema de transcripción musical basado en redes neuronales, que mediante técnicas de procesado y alineamiento de audio consigue obtener una predicción de la sincronización entre una señal original, referente a una pieza de música en concreto, con cualquier grabación realizada de la interpretación de dicha pieza.

Más adelante, se realizará una pequeña interfaz, a modo de aplicación web, para mostrar los resultados obtenidos de manera más dinámica, con la ayuda también de un software de notación musical.

Se van a abarcar tecnologías de aprendizaje automático y aprendizaje profundo basadas en redes neuronales. También, será necesario usar herramientas de procesamiento de datos para adecuar lo mejor posible las señales al sistema propuesto.

Para el desarrollo del programa de transcripción musical y su procesamiento se va a utilizar Python como lenguaje y entorno de programación, apoyándose también en Matlab.

En cuanto al desarrollo web, se tratará en este caso en HTML + CSS y JavaScript.

Además, se establecerá conexión entre la App web creada y el software de notación musical, en concreto una conexión llamada OSC, para monitorizar los resultados.

1.2 MOTIVACIÓN

1.2.1 HIPÓTESIS

La ciencia y la tecnología principalmente de la información y comunicación se encuentran en pleno apogeo y esplendor, creciendo de manera exponencial prácticamente día a día, por lo que, esto puede llevar a descubrir y desarrollar nuevas formas de trabajo o de vida, en cuanto a tecnología se refiere.

Hablando de este trabajo en concreto, si fuera posible obtener un sistema de transcripción musical, entonces podría realizarse una alineación entre el audio de una grabación de la interpretación de una pieza musical con su partitura, de manera totalmente automática.

1.2.2 OBJETIVOS

El objetivo general de proyecto será realizar un sistema final, a modo de aplicación web, capaz de mostrar en tiempo real la sincronización de cualquier señal estimada por el software de predicción creado, con su señal original asociada.

Para ello, se va a realizar la creación de una base de datos suficiente para entrenar las redes neuronales propuestas.

También, se hará una evaluación de los modelos de redes configurados, escogiendo finalmente el mejor caso para el sistema práctico final.

Además, se realizará el correspondiente procesado y alineamiento obteniendo un emparejamiento entre la señal original y la predicha por el sistema de transcripción musical, evaluando este sistema de sincronismo con varias configuraciones de señales para comprobar su funcionamiento.

Se desarrollará una interfaz web capaz de establecer comunicación con un software de notación musical para mostrar los resultados obtenidos en el sistema final, aportando dinamismo e interactividad, en cuanto a visualización estética se refiere.

2. TRANSCRIPCIÓN MUSICAL

2.1 INTRODUCCIÓN

En este trabajo se ha elaborado un software de transcripción musical basado en redes neuronales, concretamente “Deep Learning”.

Para ello, antes de todo se ha escogido un lenguaje de programación adecuado con el que llevar a cabo este proyecto. En nuestro caso se ha elegido Python.

También, ha sido de utilidad la transformada Constant-Q, con la que se han procesado las señales temporales correspondientes para obtener los coeficientes necesarios previos al entrenamiento de las redes neuronales.

Una vez obtenida la base de datos a usar ya procesada, se han implementado diversos casos de redes neuronales con los cuales se realizará una evaluación para escoger tras una comparación entre los resultados obtenidos el mejor de ellos y así de esta manera tener posteriormente una gran calidad y precisión en el sistema propuesto final.

2.2 PYTHON

2.2.1 Historia del software

Python fue creado a principios de la década de 1990 por Guido van Rossum en Stichting Mathematisch Centrum en los Países Bajos como sucesor de un lenguaje llamado ABC. Guido sigue siendo el autor principal de Python, aunque incluye muchas contribuciones de otros muchos.

En 1995, Guido continuó su trabajo en Python en la Corporación para Iniciativas Nacionales de Investigación (CNRI) en Reston, Virginia, donde lanzó varias versiones del software.

En mayo de 2000, Guido y el equipo de desarrollo central de Python se mudaron a BeOpen.com para formar el equipo de BeOpen PythonLabs. En octubre del mismo año, el equipo de PythonLabs se trasladó a Digital Creations (ahora Zope Corporation). En 2001, se formó la Python Software Foundation (PSF), una organización sin fines de lucro creada específicamente para poseer la propiedad intelectual relacionada con Python. Zope Corporation es un miembro patrocinador de la PSF.

Todas las versiones de Python son de código libre.

2.2.2 Principales características

Algunas de las características notables de Python:

Utiliza una sintaxis elegante, que facilita la lectura de los programas que son escritos.

Es un lenguaje fácil de usar que simplifica el funcionamiento de su programa. Esto hace que Python sea ideal para el desarrollo de prototipos y otras tareas de programación ad-hoc, sin comprometer la capacidad de mantenimiento.

Viene con una gran biblioteca estándar que admite muchas tareas de programación comunes, como conectarse a servidores web, buscar texto con expresiones regulares, leer y modificar archivos, entre muchas otras.

El modo interactivo de Python facilita la prueba de fragmentos cortos de código. También hay un entorno de desarrollo incluido llamado IDLE.

Se extiende fácilmente agregando nuevos módulos implementados en un lenguaje compilado como C o C ++.

También se puede incrustar en una aplicación para proporcionar una interfaz programable.

Funciona en cualquier lugar, incluyendo MacOS X, Windows, Linux y Unix, con versiones no oficiales también disponibles para Android y iOS.

Es software libre en dos sentidos, es decir, no cuesta nada descargar o usar Python, o incluirlo en su aplicación y además Python también se puede modificar y redistribuir libremente, porque aunque el lenguaje tiene derechos de autor, está disponible bajo una licencia de código abierto .

2.2.3 ¿Por qué Python para este proyecto?

Este proyecto se puede decir que trata principalmente sobre Machine Learning o más concretamente sobre Deep Learning en cuanto a programación se refiere.

Deep Learning, en términos simples, es utilizar los datos para hacer que una máquina tome una decisión inteligente. Por ejemplo, se puede crear un algoritmo de detección de correo no deseado en el que se puedan aprender las reglas a partir de los datos o una detección anómala de eventos raros al consultar datos anteriores u organizar su correo electrónico según las etiquetas que haya asignado al aprender sobre el historial de correo electrónico, etc.

Por tanto, esto no es más que reconocer patrones en la cantidad de datos que sea posible para trabajar.

Una tarea importante de un ingeniero o programador que trabaja en Machine Learning en su vida laboral es extraer, procesar, definir, limpiar, organizar y luego comprender los datos para desarrollar algoritmos inteligentes.

Debido a todas estas tareas, que son contempladas en el proyecto que a continuación se expone, se ha escogido Python, ya que, como se puede ver en sus características principales es un lenguaje fácil y rápido de comprender y aunque los cálculos matemáticos, álgebra,

procesado, entre otros, sean bastante complejos, Python permite realizar una implementación de manera sencilla y muy entendible que ayuda a aclarar ideas en diversas situaciones.

Los datos son la clave, por lo tanto, depende totalmente del tipo de tarea en la que desee aplicar Aprendizaje automático. Los datos de entrada pueden ser tanto una imagen o video como una serie de puntos a lo largo del tiempo, una recopilación de documentos de idiomas distribuidos en varios dominios, archivos de audio o solo algunos números. Toda esa posible cantidad de datos, a priori, son en la mayoría de los casos solo datos en crudo, es decir, datos desestructurados, desajustados, incompletos, desnormalizados, etc.

Para ser bien tratados y arreglados para su posterior uso, Python posee una gran variabilidad de paquetes y librerías a plena disposición en sus repositorios de código abierto en continuo desarrollo y mejora.

El único inconveniente o contra por la que Python nunca es o será muy usado hasta que no sean mejorados aspectos técnicos computacionales es debido a la sobrecarga que conlleva. Pero para aclarar el caso, nunca se creó para el sistema sino para la usabilidad. Los procesadores pequeños o el hardware con poca memoria no se adaptan a la base de código de Python en la actualidad, pero para estos casos se tiene C y C ++ como principales herramientas de desarrollo.

2.3 CONSTANT-Q TRANSFORM

La transformada Constant-Q se trata de una herramienta matemática muy ligada a la común transformada de Fourier en procesamiento de audio. Al igual que la transformada de Fourier, la transformada Constant-Q se compone de un llamado banco de filtros, pero que en contraste con esta otra transformada, la sucesión de filtros en este caso posee el centro de sus frecuencias espaciadas de manera geométrica:

$$f_k = f_0 \cdot 2^{\frac{k}{b}} \ (k = 0,1,\dots) \quad \textbf{(1)}$$

donde b se refiere al número de filtros por cada octava.

Para conseguir que los filtros no se solapen y cubran todo el rango de frecuencias esperado se elige el ancho de banda del filtro k-ésimo como:

$$\Delta_k^{cq} = f_{k+1} - f_k = f_k \cdot (2^{\frac{1}{b}} - 1) \quad (2)$$

Esto produce una relación constante entre las frecuencias centrales consecutivas con una resolución Q expresada como:

$$Q = \frac{f_k}{\Delta_k^{cq}} = (2^{\frac{1}{b}} - 1)^{-1} \quad \textbf{(3)}$$

Lo que hace que la transformada Constant-Q tan útil es el hecho de que una buena elección en la frecuencia central mínima (f_0) y del número de filtros por octava (b) hace que las k-ésimas frecuencias centrales se correspondan directamente con sus respectivas notas musicales. Como ejemplo, eligiendo b = 12 y f_0 como la frecuencia de la nota 0 en formato MIDI, hacen que el resto de frecuencias centrales tengan su pareja en la k-ésima nota MIDI.

Otra característica para resaltar de dicha transformada se trata de la creciente resolución temporal hacia altas frecuencias.

Esta situación tiene una gran relación con el sistema auditivo humano. No solo ocurre en la computación digital la necesidad de un tiempo para percibir la frecuencia de un tono grave si no que también ocurre en el sentido auditivo. Esto esta relacionado a la música, frecuentemente siendo más afín en registros bajos.

En este proyecto se ha usado una librería de Python llamada "librosa". Ésta es muy útil para cualquier tipo de procesamiento de audio, por tanto, contiene los módulos necesarios para realizar la transformada Constant-Q a las piezas de audio que se requieran.

Para ello, "librosa.cqt()" necesita principalmente 5 parámetros de entrada para devolver en su salida los coeficientes transformados listos para su entrenamiento posterior en la red de aprendizaje profundo.

Éstos deben ser:

- Señal temporal de entrada a procesar.

- Frecuencia de muestreo (fs). En este caso concreto se ha escogido 44100Hz.

- Hop_length. Este parámetro consiste en la relación de muestras temporales que se procesan por cada trama espectral resultante. En este caso se ha usado el valor por defecto, siendo este 512, es decir, por cada 512 muestras de la señal de entrada se ha obtenido 1 trama en el dominio transformado.

- Bins_per_octave. Como se ha dicho previamente, para la configuración de la transformada es necesario indicar el número de filtros por octava o lo que es lo mismo, la variable b en las ecuaciones anteriores. En este proyecto se ha establecido un valor constante de 36 filtros por octava.

- N_bins. Se trata del número de coeficientes que componen cada una de las tramas espectrales. Se puede calcular como el número de octavas a evaluar por el número

de filtros que contiene cada octava (bins_per_octave). Por lo que, al querer utilizar para este proyecto 7 octavas teniendo cada una de ellas 36 filtros configurados, se obtiene un total de 252 coeficientes por cada trama, siendo cada trama distinta de cualquier otra, debido a que este vector se va modelando según las características de cada una de las 512 muestras de la señal de entrada correspondiente a cada una de las tramas.

Estos coeficientes serán los responsables principales del redimensionamiento de la señal de entrada, pasando esta de ser un vector de muestras a lo largo del tiempo, a ser una matriz formada por el número de tramas en un eje y los N_bins, o lo que es lo mismo, los coeficientes transformados por cada una de esas 512 muestras temporales. Es decir, esta dimensión es independiente de todo lo demás y siempre será igual a 252.

Otro de los parámetros constantes e independientes será el número de notas MIDI correspondiente con las demás variables establecidas. En este caso, al tener 7 octavas se obtendrá un total de 88 notas, yendo estas desde la nota MIDI 21 hasta la 108, ya que es el rango medio para la mayoría de las composiciones de piano existentes.

2.4 TRANSCRIPCIÓN USANDO REDES NEURONALES

2.4.1 REDES NEURONALES

2.4.1.1 INTRODUCCIÓN

Las redes neuronales son uno de los paradigmas de programación más interesantes jamás inventados. En el enfoque convencional de la programación, le decimos a la computadora qué hacer, dividiendo grandes problemas en muchas tareas pequeñas y definidas con precisión que la computadora puede realizar fácilmente. Por el contrario, en una red neuronal no le decimos a la computadora cómo resolver el problema. En cambio, aprende de los datos de observación, descubriendo su propia solución al problema en cuestión.

Las redes neuronales más usadas en la actualidad se basan en lo que se conoce como aprendizaje supervisado.

2.4.1.2 APRENDIZAJE SUPERVISADO

Este aprendizaje supervisado se basa en conocer, a priori, las variables de entrada y también la o las de salida y, por tanto, usar un algoritmo para aprender la función de mapeo que se recorre desde la capa de entrada a la salida.

$$Y = f(X) \quad (4)$$

El objetivo es aproximar tan bien como sea posible la función de mapeo de modo que cuando se le introduzcan nuevos datos de entrada (X) a dicha función, esta sea capaz de predecir la variable de salida (Y) correspondiente estos datos de entrada con el mínimo error cometido.

Se llama aprendizaje supervisado porque el proceso de un algoritmo que aprende del conjunto de datos de capacitación puede considerarse como un maestro que supervisa el proceso de aprendizaje. El programa conoce las respuestas correctas, el algoritmo realiza predicciones de forma iterativa sobre los datos de entrenamiento y va siendo corregido por el profesor. El aprendizaje se detiene cuando el algoritmo cree alcanzar un nivel adecuado y aceptable en el rendimiento del problema.

Los problemas de aprendizaje supervisados pueden agruparse en problemas de regresión y clasificación.

- **Clasificación**: Un problema de clasificación es cuando la variable de salida es una categoría, como "rojo" o "azul" o "enfermedad" y "sin enfermedad".

- **Regresión**: Un problema de regresión es cuando la variable de salida es un valor real, como "dólares" o "peso".

Para ambos problemas de aprendizaje automático una parte muy importante a tener en cuenta son los datos de entrada. Estos datos según su estructuración pueden ser:

- **Datos estructurados:** Los datos estructurados son datos que se adhieren a un modelo de datos predefinido y, por lo tanto, son fáciles de analizar. Los datos estructurados se ajustan a un formato tabular con relación entre las diferentes filas y columnas. Ejemplos comunes de datos estructurados son archivos Excel o bases de datos SQL. Cada uno de estos tiene filas y columnas estructuradas que se pueden ordenar.

  Los datos estructurados dependen de la existencia de un modelo de datos, un modelo de cómo se pueden almacenar, procesar y acceder a los datos. Debido a un modelo de datos, cada campo es discreto y se puede acceder por separado o en conjunto, junto con los datos de otros campos. Esto hace que los datos estructurados sean extremadamente poderosos.

- **Datos no estructurados:** Los datos no estructurados son información que no tiene un modelo de datos predefinido o que no está organizada de manera predefinida. La información no estructurada suele contener texto, pero también puede contener datos como fechas, números y hechos. Esto da como resultado irregularidades y ambigüedades que dificultan la comprensión del uso de programas tradicionales en comparación con los datos almacenados en bases de datos estructuradas. Los ejemplos comunes de datos no estructurados incluyen archivos de audio, video o bases de datos No-SQL.

La capacidad de almacenar y procesar datos no estructurados ha crecido enormemente en los últimos años. Este hecho es especialmente relevante en el avance de estas tecnologías emergentes, ya que, en gran parte la mayoría de los datos existentes no están bien organizados o estructurados.

En concreto, en este proyecto se van a usar especialmente una serie de base de datos no estructurada que se mostrará en puntos posteriores.

Para llevar a cabo un problema con aprendizaje supervisado, a parte de tener los datos necesarios se debe de buscar una óptima arquitectura de red, la cual depende de la aplicación en cuestión con la que se vaya a trabajar.

A groso modo, se puede decir que existen 3 tipos de redes principales más usados:

- **Red de perceptrones multicapa:** es posiblemente la más simple de las 3, pero también por ello la más usada, debido a sus muy buenos resultados en muchos de los principales problemas propuestos en la actualidad incluso teniendo un coste computacional mucho inferior a los otros dos tipos de arquitecturas neuronales.

  Por ello, será la principal tipología de red usada en este proyecto, aunque también se mostrará en pequeño detalle algún ejemplo de la red convolucional.

- **Red convolucional**: las redes neuronales convolucionales son una técnica poderosa de redes neuronales artificiales. Estas redes preservan la estructura espacial del problema y se desarrollaron principalmente para tareas de reconocimiento de objetos como el reconocimiento de dígitos escritos a mano. Son populares porque las personas están logrando resultados de vanguardia en tareas difíciles de visión por computadora y procesamiento de lenguaje natural.

  Este tipo de redes también pueden ser llamadas ConvNets o CNN.

- **Red recurrente:** existe otro tipo de red neuronal que está dominando los problemas difíciles de aprendizaje automático que involucran secuencias de entradas. Las redes neuronales recurrentes tienen conexiones formadas por bucles, lo que agrega retroalimentación y memoria a las redes con el tiempo. Esta memoria permite que este tipo de red aprenda y generalice a través de secuencias de entradas en lugar de patrones individuales. Se ha demostrado que un tipo poderoso de red neuronal recurrente llamada red de memoria a corto plazo (en inglés, Long short-term memory network o LSTM) es particularmente efectiva cuando se apila en una configuración profunda, logrando resultados de última generación en una amplia gama de problemas, desde la traducción de idiomas hasta los subtítulos automáticos de imágenes y videos.

Sabiendo esto, se puede decir que este proyecto trata sobre un algoritmo de clasificación múltiple binaria a partir de datos no estructurados, por lo que, se va a explicar a continuación como aplicarlo en redes de perceptrones multicapa principalmente, pero también en redes convolucionales. Para ello, se detalla en los puntos siguientes como se configuran dichas redes en el caso en cuestión.

2.4.1.3 RED MLP (Perceptrón Multicapa)

¿Qué es una red neuronal de perceptrones multicapa?

Para comenzar, se va a explicar un tipo de neurona artificial llamada *perceptrón*.

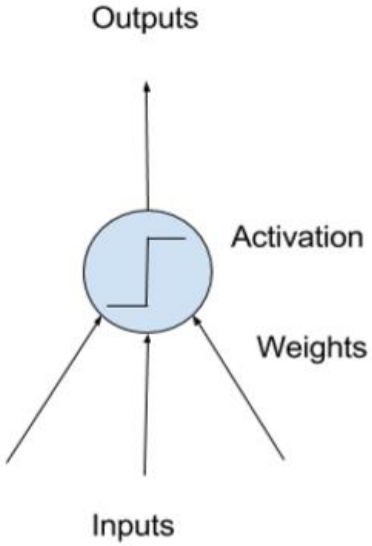
Los perceptrones fueron desarrollados en las décadas de 1950 y 1960 por el científico Frank Rosenblatt , inspirado en trabajos anteriores de Warren McCullochy Walter Pitts .

A día de hoy, existen diversos tipos de neuronas más usadas que el perceptrón. Éstas serán citadas más adelante, pero para su buen entendimiento vale la pena primero tomar un tiempo para entender que son los perceptrones.

Perceptrón es un modelo de neurona única que fue un precursor de redes neuronales más grandes. Es un campo de estudio que investiga cómo se pueden utilizar modelos simples del cerebro biológico para resolver tareas computacionales difíciles como las tareas de modelado predictivo que vemos en el aprendizaje automático. El objetivo no es crear modelos realistas del cerebro, sino desarrollar algoritmos robustos y estructuras de datos que se puedan utilizar para modelar problemas difíciles.

Estos perceptrones se tratan de unidades computacionales simples que ponderan señales de entrada y producen una señal de salida mediante una función de activación.

Ilustración i. Arquitectura básica de un perceptrón



En la imagen se puede ver 3 entradas, pero en general suelen tratarse de unas pocas más o muchas más, dependiendo del problema a cubrir.

Estas entradas son ponderadas mediante unos pesos propios de cada aplicación llamados "weights" (del inglés). Estos pesos son números reales que expresan la importancia de dichas entradas con respecto a cada salida.

Cada perceptrón produce una salida simple binaria, ya sea 0 ó 1, la cual determina si la suma de las entradas ponderadas sobrepasa o no un cierto umbral. En términos algebraicos:

$$output = \begin{cases} 0 & if \ \sum_j w_j x_j \leq threshold \\ 1 & if \ \sum_j w_j x_j > threshold \end{cases} \quad (5)$$

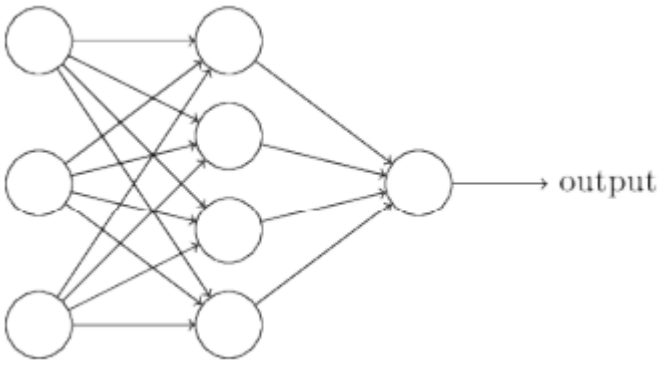
Tanto los "weights" como el valor umbral, llamado "threshold" (del inglés) son parámetros del perceptrón que pueden ser variados dependiendo del modelo de decisión que se quiera tomar.

Arquitectura de la red neuronal.

El poder de las redes neuronales proviene de su capacidad para aprender la representación en los datos de entrenamiento y cómo relacionarlo mejor con la variable de salida que desea predecir. En este sentido, las redes neuronales aprenden un mapeo.

Matemáticamente, son capaces de aprender cualquier función de mapeo y se ha demostrado que es un algoritmo de aproximación universal. La capacidad predictiva de las redes neuronales proviene de la estructura jerárquica o multicapa de las redes.

Ilustración ii. Red neuronal básica



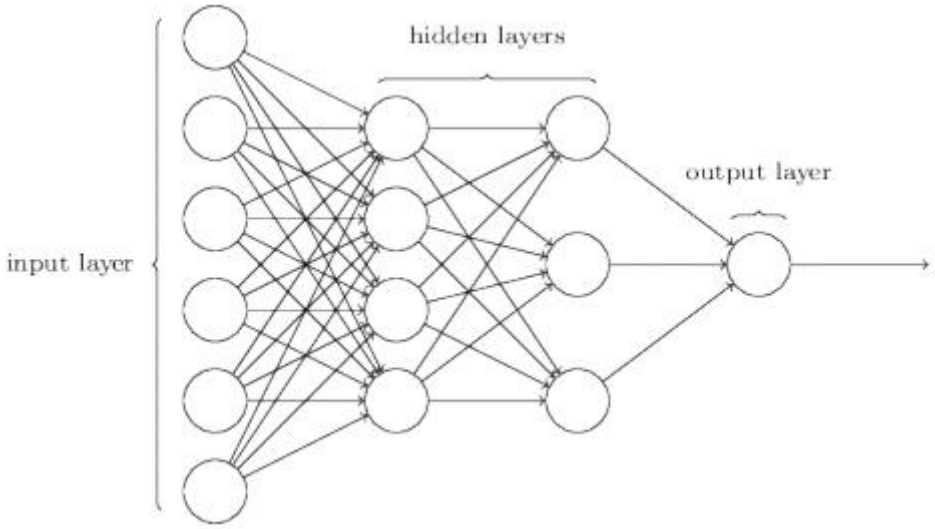
Como se ha mencionado anteriormente la capa más a la izquierda se le llama capa de entrada y, por tanto, las neuronas internas de dicha capa se conocen como "neuronas de entrada".

Por otro lado, la capa a la derecha del todo o capa de salida contiene las neuronas de salida, valga la redundancia. En este caso se puede apreciar una única neurona en esta capa, ya que puede tratarse de una red neuronal que clasifica únicamente una respuesta binaria simple.

La capa interior o de en medio es conocida como la capa oculta (en inglés, "hidden layer"), debido a que las neuronas de esta no se corresponden con entradas ni salidas.

La red de la imagen expuesta anteriormente solamente posee una capa oculta, pero normalmente en la mayoría de redes más usadas el número de capas interiores es mayor.

Ilustración iii. Red neuronal



Algo realmente confuso en redes neuronales es el hecho de llamar a diversos tipos de redes compuestas por varias capas ocultas "perceptrones multicapa" o más coloquialmente "MLPs" (siglas en inglés), ya que precisamente estas en la mayoría de los casos no se componen de perceptrones, si no, de otros tipos de neuronas, por ejemplo, en este proyecto la red MLPs propuesta se forma de neuronas RELU y neuronas sigmoideas.

Normalmente, el nombre atribuido a cada tipo de neurona se debe a la función de activación que se ejecuta en ella.

Funciones de activación

Las funciones de activación son un parámetro a configurar extremadamente importante de las redes neuronales artificiales. Básicamente deciden si una neurona debe activarse o no, si la información que está recibiendo la neurona es relevante para la información dada o debe ser ignorada.

En definición es la transformación no lineal que hacemos sobre la señal de entrada. Esta salida transformada se envía a la siguiente capa de neuronas como entrada.

$$Y = g(z), siendo\ z = wx + b\ , \quad (6)$$

donde w son los pesos de cada neurona y b el bias.

Cuando se tiene la función de activación, los pesos y el bias simplemente harían una transformación lineal. Una ecuación lineal es fácil de resolver, pero tiene una capacidad limitada para resolver problemas complejos.

Una red neuronal sin una función de activación es esencialmente solo un modelo de regresión lineal. La función de activación realiza la transformación no lineal a la entrada, lo que la hace capaz de aprender y realizar tareas más complejas. Se debería conseguir que las redes neuronales trabajen en tareas complicadas como traducciones de idiomas y clasificaciones de imágenes. Las transformaciones lineales nunca podrían realizar tales tareas.

Además, las funciones de activación hacen posible la propagación hacia atrás ya que los gradientes se suministran junto con el error para actualizar los pesos y los bias. Sin la función no lineal diferenciable, esto no sería posible.

Por ello, existen diversos tipos de funciones no lineales, dependiendo de la aplicación de la red se pueden usar unas u otras para un mejor resultado.

En este caso se han utilizado posiblemente los dos tipos más comunes en problemas de aprendizaje automático. Estos son:

- **Funciones Sigmoides.**

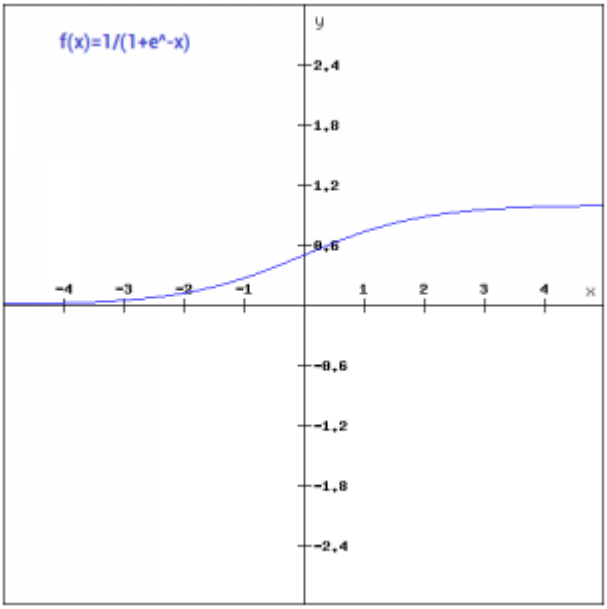
  La función de activación es una de las funciones base primarias en cuanto a aprendizaje automático se refiere y, por tanto, ampliamente usada en este tipo de aplicaciones aún a día de hoy.

  Si se traza la función siendo esta, según su definición:

$$f(x) = \frac{1}{1+e^{-x}} \quad (7)$$

Se puede representar como:

Ilustración iv. Función de activación Sigmoide



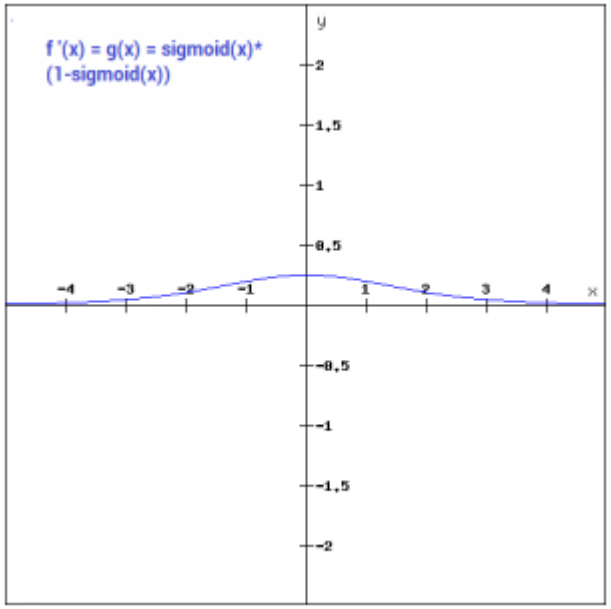
Como se aprecia, se trata de una función de pendiente suave, continua y diferenciable.

Esta función como se ha dicho previamente es una función no lineal, por lo que, la salida de las neuronas que contengan este tipo de activación tampoco será lineal.

Debido a que la curva varía de 0 a 1 en forma de "S" y que el gradiente es máximo en los valores más cercanos a 0 de la entrada y bastante más plano en regiones lejanas al eje de ordenadas, esto puede provocar que pequeños cambios de la variable de entrada en la zona de mayor gradiente se conviertan en grandes cambios en la salida, es decir, la función esencialmente trata de empujar los valores de salida hacia los extremos. Esta es una cualidad muy deseable, ya que, lo que se pretende es clasificar los valores de entrada.

Representando y definiendo también la forma del gradiente de la función sigmoide:

Ilustración v. Gradiente de la función de activación Sigmoide



Siendo: $$f'(x) = f(x) \cdot (1 - f(x)) = \frac{1}{1+e^{-x}} \cdot (1 - \frac{1}{1+e^{-x}}) \quad (8)$$

Se puede decir que depende del valor de entrada, sobre todo, con mayor fuerza en valor cercanos a 0. Esto significa que durante la propagación hacia atrás podemos usar fácilmente esta función. El error se puede propagar hacia atrás y los pesos se pueden actualizar en consecuencia.

Por otro lado, debido a la característica de gradiente plano en regiones mas lejanas al origen puede surgir un problema. Esto significa que una vez que la función cae en esa región, los gradientes se vuelven muy pequeños. Esto significa que el gradiente se acerca a cero y la red no está realmente aprendiendo.

Otro problema que sufre la función sigmoide es que los valores solo oscilan entre 0 y 1. Esto significa que la función sigmoide no es simétrica alrededor del origen y los valores recibidos son todos positivos. Por lo que, no siempre es deseable que los valores que van a la siguiente neurona sean todos del mismo signo. Esto se puede solucionar escalando la función sigmoide.

Para ello, surge la función “Tanh”.

Realmente, es solo una versión escalada de la función sigmoide, pudiendo definirse a partir de ella como:

$$tanh(x) = 2 \cdot sigmoid(2x) - 1 \quad (9)$$

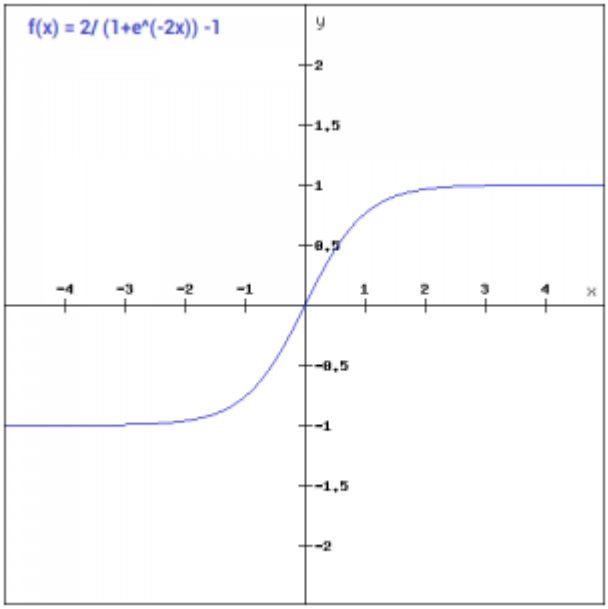
siendo *sigmoid(x)* la función sigmoide vista justo previamente.

También puede escribirse directamente como:

$$tanh(x) = \frac{2}{1+e^{-2x}} - 1 \quad (10)$$

Tanh funciona de manera similar a la función sigmoide, pero es simétrica sobre el origen. Su imagen oscila entre -1 y 1.

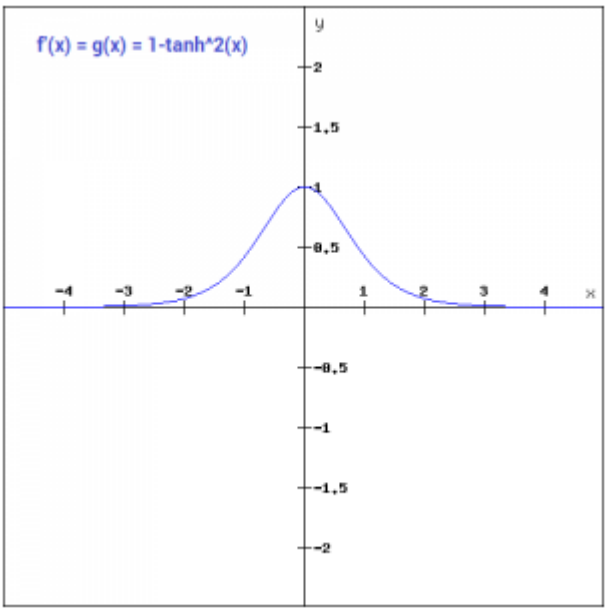
Ilustración vi. Función de activación Tanh



Básicamente resuelve el problema de que todos los valores sean del mismo signo. Todas las demás propiedades son las mismas que las de la función sigmoidea. Es continua y diferenciable en todos los puntos. La función, como se puede ver, no es lineal, por lo que es posible propagar los errores fácilmente.

El gradiente de la función Tanh es más pronunciado en comparación con la función sigmoide. La elección de usar sigmoide o Tanh dependería básicamente del requisito de gradiente en la declaración del problema. Pero de manera similar a la función sigmoidea, todavía puede aparecer el problema del gradiente de fuga. La gráfica de la función Tanh es plana y los gradientes son muy bajos fuera de la región adyacente al origen. Como se puede observar en la figura de abajo:

Ilustración vii. Gradiente de la función de activación Tanh



Siendo, $$f'(x) = 1 - tanh^2(x) \quad (11)$$

❖ **Funciones ReLU.**

Se trata de una unidad lineal rectificada (en inglés, rectified linear unit).

ReLU es la función de activación más utilizada al diseñar redes en la actualidad. Primeramente, la función ReLU no es lineal, lo que significa que podemos propagar fácilmente los errores y tener múltiples capas de neuronas activadas por la función ReLU.

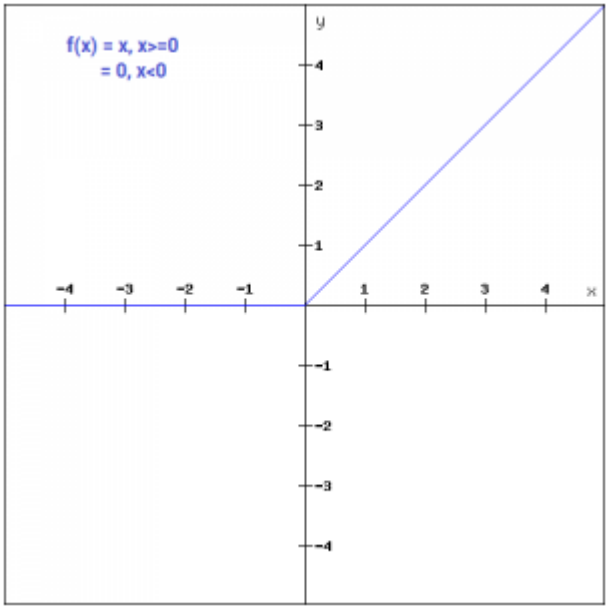
Esta función puede definirse como:

$$f(x) = max(0, x) \quad (12)$$

es decir, cuando los valores de entrada están por debajo del umbral 0, la neurona permanecerá desactivada y cuando la entrada es positiva, la función se comporta como una función lineal constante.

Se puede representar gráficamente como:

Ilustración viii. Función de activación ReLU



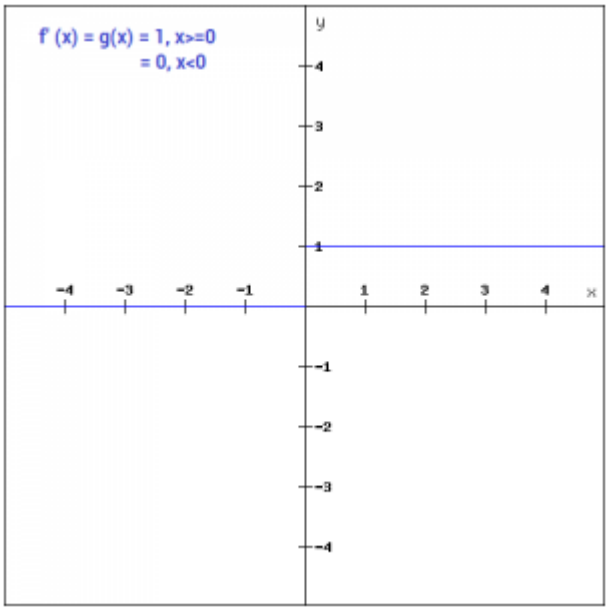
La principal ventaja de usar la función ReLU sobre otras funciones de activación es que no activa todas las neuronas al mismo tiempo. ¿Qué significa esto? Como se ha dicho anteriormente, si la entrada es negativa la convertirá a cero y la neurona no se activará. Esto significa que en un momento solo se activan unas pocas neuronas, lo que hace que la red sea escasa, por lo que la hace eficiente y fácil para el cálculo.

Por otro lado, el problema es que todos los valores negativos se vuelven cero inmediatamente, lo que disminuye la capacidad del modelo para ajustarse o entrenarse a partir de los datos correctamente.

Esto viene relacionado con los gradientes que se mueven hacia cero.

Si se observa el lado negativo del gráfico del gradiente o función derivada de ReLU:

Ilustración ix. Gradiente de la función de activación ReLU



Siendo**:**

$$f'(x) = \begin{cases} 0, & x < 0 \\ 1, & x \geq 0 \end{cases} \qquad (13)$$

El gradiente es cero, lo que significa que, para activaciones en esa región, los pesos no se actualizan durante la propagación inversa, que será descrita más adelante. Esto puede crear neuronas muertas que nunca se activarán.

Para intentar corregir el problema de escasa capacidad de ajuste surge una función derivada de ReLU, llamada Leaky ReLU o ReLU con fugas.

La función ReLU con fugas no es más que una versión mejorada de la función ReLU.

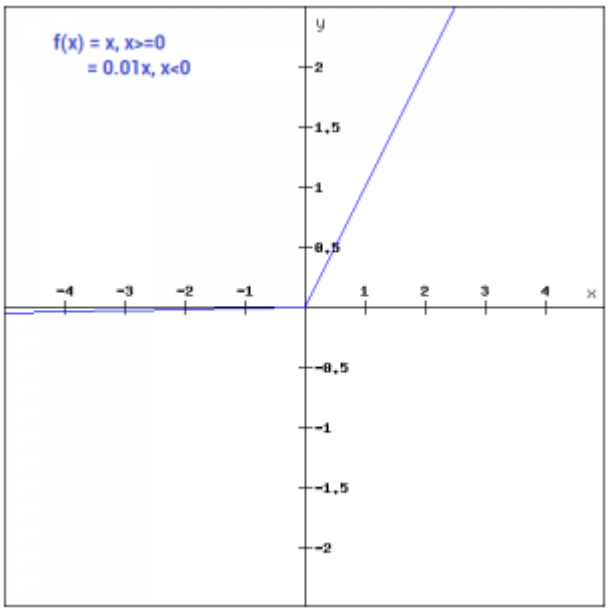
En lugar de definir la función como 0 para x menor que 0, la definimos como un pequeño componente lineal de x.

$$f(x) = \begin{cases} ax, & x < 0 \\ x, & x \geq 0 \end{cases} \qquad (14)$$

Siendo *a* un valor predeterminado normalmente cercano a cero, por ejemplo, *a = 0.01*.

Aquí simplemente se ha reemplazado la línea horizontal con una línea no horizontal y no nula. Se puede representar como:

Ilustración x. Función de activación Leaky ReLU

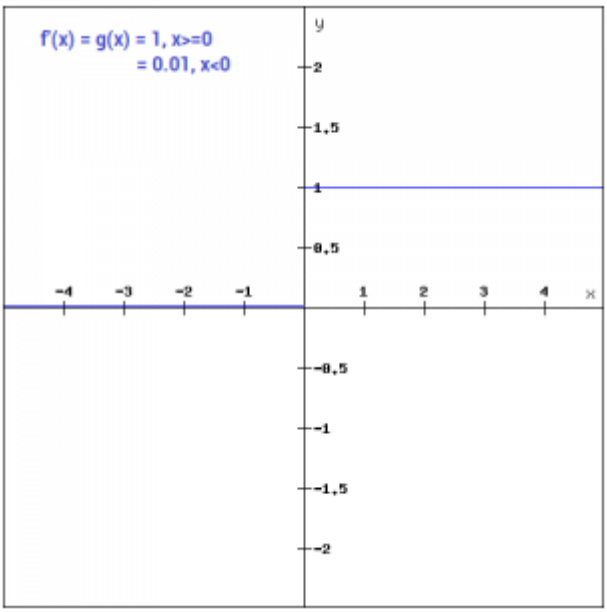


La principal ventaja de reemplazar la línea horizontal es eliminar el gradiente nulo.

Entonces, en este caso, el gradiente de la izquierda del eje de ordenadas al no ser cero, ya no se encontrarían neuronas muertas en esa región.

La gráfica de la función derivada se muestra de la siguiente manera:

Ilustración xi. Gradiente de la función de activación Leaky ReLU



Siendo: $$f'(x) = \begin{cases} a, & x < 0 \\ 1, & x \geq 0 \end{cases} \quad (15)$$

Donde *a* es un valor no nulo cercano a cero.

Otra función de activación bastante usada en aplicaciones de clasificación principalmente y muy ligada a la función sigmoide, se trata de la función **Softmax**.

A diferencia de la activación sigmoide que solo es capaz de tratar 2 clases distintas, Softmax surge para incluir la posibilidad de clasificación de múltiples clases simultáneamente.

La función Softmax se usa idealmente en la capa de salida del clasificador donde realmente se trata de alcanzar las probabilidades para definir la clase de cada entrada.

Una vez conocido esto, a groso modo,

¿cuál es la función de activación correcta dependiendo de cada caso?

No existe una regla general para escoger una u otra, sin embargo, según diversas condiciones del problema se puede hacer una mejor elección para lograr una convergencia más rápida y simple de la red.

A modo de ejemplo, se puede decir que, en resumen:

- Las funciones sigmoides y sus combinaciones generalmente funcionan mejor en el caso de los clasificadores.
- Las funciones sigmoides y Tanh a veces se evitan debido al problema del gradiente de fuga.
- La función ReLU es una función de activación general y se usa en la mayoría de los casos.
- Siempre hay que tener en cuenta que la función ReLU solo debe usarse en las capas ocultas.
- Como regla general, se debe comenzar usando la función ReLU y luego pasar a otras funciones de activación en caso de que ReLU no proporcione resultados óptimos.

Conociendo todo esto y sabiendo que la aplicación propuesta en este proyecto se trata de un algoritmo de clasificación múltiple binaria se explicará en los siguientes puntos como se desarrolla este proceso de clasificación con redes de aprendizaje profundo de manera analítica.

2.4.1.4 Red neuronal profunda (DNN)

Introducción

Se trata de un tipo de redes basadas previamente en redes MLP, pero con una cantidad de unidades y capas muy superior, siendo capaz de aprender mucho más a fondo, eso sí, necesitando una cantidad de datos e información muy superior, ya que, si no la red no sería capaz seguramente de aprender de manera óptima.

Los pasos principales para construir una red neuronal de aprendizaje profundo son:

- Definir la estructura del modelo (como el número de características de entrada y salidas)
- Inicializar los parámetros del modelo.
- Bucle.
    - Calcular la pérdida actual (forward propagation)
    - Calcular gradiente actual (back propagation)
    - Parámetros de actualización (descenso de gradiente)
- El preprocesamiento del conjunto de datos es importante.
- Ajustar la tasa de aprendizaje (que es un ejemplo de un "hiperparámetro") puede marcar una gran diferencia en el algoritmo.

Según lo explicado previamente, esos conceptos básicos ayudarán a una mejor comprensión de lo siguiente:

Cuando una persona empieza a trabajar, a probar, a implementar sus primeros problemas de aprendizaje automático o más concretamente profundo, se debe preguntar cómo es el proceso y el camino que siguen los datos de entrada y sus correspondientes salidas para que la red aprenda y sea capaz de predecir una vez que se ha entrenado dicha red.

Bien, pues principalmente de manera básica en redes multicapa se siguen dos procesos básicos para el aprendizaje de redes neuronales.

- **PROPAGACION HACIA ADELANTE (FORWARD PROPAGATION)**
- **PROPAGACION HACIA ATRÁS (BACKWARD PROPAGATION)**

Antes de explicar en qué consisten ambos procesos, es bueno ampliar un poco con conceptos básicos fundamentales de cómo funciona una red internamente.

Para ello, debido a que, como se ha expuesto antes, se va a tratar de forma más detallada el problema de redes destinadas a clasificación binaria y como se ha dicho anteriormente para ello es necesario un algoritmo basado en aprendizaje supervisado.

Un clasificador binario consiste principalmente en predecir, a partir de unos valores de entrada dados, cuál será el valor de salida final más probable siguiendo una serie de parámetros.

Cada ejemplo de clasificación que pueda ser propuesto se representa mediante, una entrada que se le llama:

$$x,\ donde\ x \in \mathbb{R}^{nx} \quad (16)$$

Esto significa básicamente que *x* es un vector de números reales asociados a características concretas del problema que tiene una dimensión de n_x.

Por otro lado, se tiene la variable de salida del clasificador que como al ser binario se dividen en solo dos clases de la forma:

$$y \in (1,0)$$

Pudiendo la salida tomar o valor 1 o valor 0 según la clasificación.

A modo de ejemplo, si se intenta predecir si una persona es feliz o triste, donde estar feliz se da si $y = 0$ y triste si $y = 1$. Suponiendo que las características de entrada dadas al problema pueden ser:

- Cuanto duerme esa persona al día.
- Cuanto ejercicio realiza dicha persona semanalmente.
- Cuanta vida social realiza a la semana.

Como se aprecia a simple vista, existen 3 parámetros para evaluar al sujeto, por lo que, cada vector de entrada de cada entrenamiento será en vector tridimensional.

Los valores de cada uno de los parámetros a tener en cuenta se van a indicar con un subíndice, en este caso (x_1, x_2, x_3).

De otro modo, se establece la variable *m*, como el número de ejemplos de entrenamiento que se realizarán en la red por lo que se obtendrán al final *m* pares de ejemplos de entrenamiento.

Para denotar cada número de entrenamientos realizados se usa un super índice *(i)* para diferenciarlo del número de parámetro que se acaba de explicar. Por lo que, de manera general, el número de ejemplos de entrenamiento i-ésimos será de la forma:

$$(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), (x^{(3)}, y^{(3)}), \ldots, (x^{(m)}, y^{(m)}) \quad (17)$$

Lo siguiente, será agrupar en una matriz $\boldsymbol{X}$ tanto los vectores formados por cada una de los parámetros a tener en cuenta, como cada uno de los i-ésimos entrenamientos realizados, por lo que, se escribe como:

$$X = \begin{bmatrix} \vdots & \vdots & \cdots & \vdots \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ \vdots & \vdots & \cdots & \vdots \end{bmatrix} \quad (18)$$

La forma de X se va a determinar por:

$$X \in \mathbb{R}^{n_x,\ m} \quad (19)$$

Es decir, la matriz tendrá tantas filas como parámetros distintos y tantas columnas como número de entrenamientos realizados.

Similarmente, también se va a agrupar todos los valores de salida *y*, para ejemplo de entrenamiento en un vector:

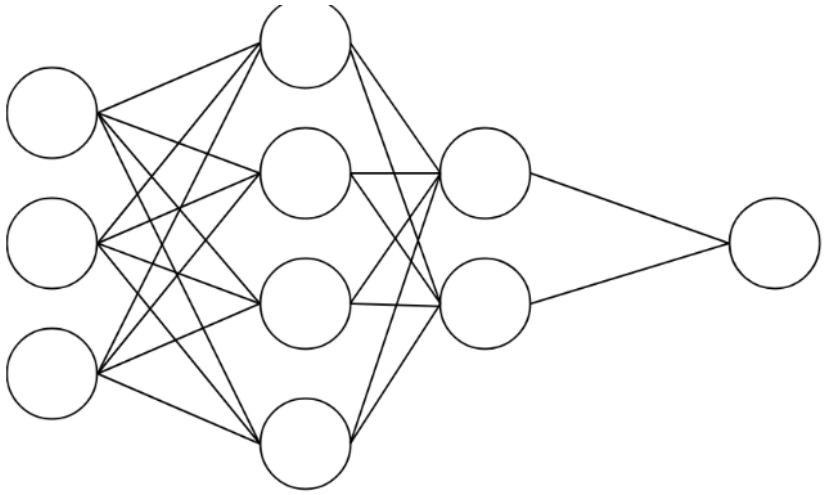
$$\boldsymbol{Y} = [y^{(1)} \quad y^{(2)} \quad \cdots \quad y^{(m)}] \quad (20)$$

La forma de Y será:

$$Y \in \mathbb{R}^{1,m} \quad (21)$$

Para explicar estos métodos de propagación se va a usar una red básica general de 3 capas. Una capa de entrada, dos capas ocultas con activación ReLU, explicado en puntos previos. Y una capa de salida que viene interconectada con la última capa oculta mediante una función de activación sigmoide.

Ilustración xii. Red neuronal general de 3 capas



En la red de la figura puede parecer como si hubiera 4 capas, pero realmente no es así.

Esto es porque la primera columna de neuronas, no es precisamente una capa de neuronas si no que corresponde con los parámetros de entrada al clasificador y se representa de dicha forma.

Se ha escogido este número de capas para asemejarlo lo máximo posible al algoritmo del proyecto final, el cual también consiste en esta misma configuración de capas, aunque obviamente con mayor cantidad de neuronas por capa.

Para empezar, la primera cuestión probablemente a abordar será que representa realmente la salida del clasificador.

Sabiendo previamente como ya se ha explicado, que la entrada $(\boldsymbol{x}^{(i)})$ se corresponde con un vector que incluye los parámetros necesarios para caracterizar el problema en cada uno de los ejemplos de entrenamiento, entonces la salida de la red será la predicción que tome esta según la configuración y los pasos que vayan sucediéndose en el entrenamiento.

Esta predicción de la salida toma siempre valores entre 0 y 1. Puede expresarse como:

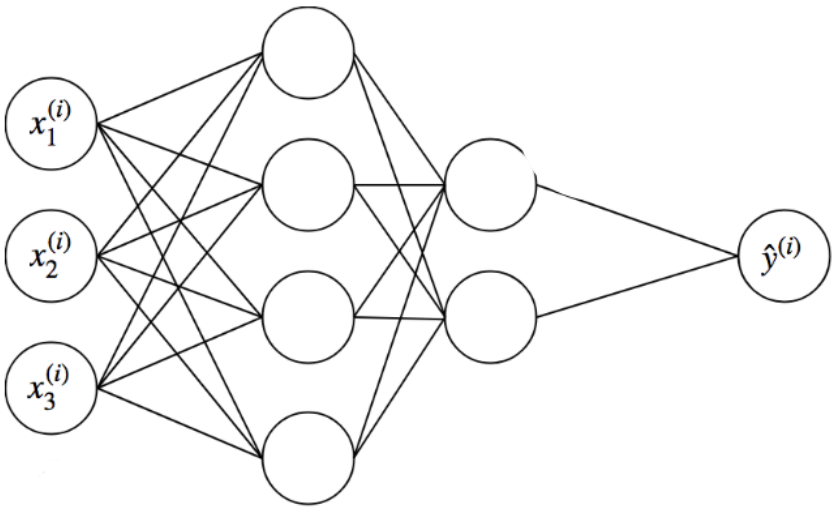
$$\hat{y} = P(y = 1 \mid x) \qquad (22)$$

Siendo $\hat{y}$ la probabilidad de que la salida de la red sea igual a 1 dado el vector de entrada en cada ejemplo de entrenamiento.

Por tanto, se define la primera capa de la red neuronal como un vector de características de un ejemplo de entrenamiento dado *(i)*. En el diagrama de abajo, cada entrada en el vector de características representa un valor escalar. Por ejemplo, $\boldsymbol{x_1^{(i)}}$ es el valor de la primera función para el i-ésimo ejemplo de entrenamiento.

Se observa que en la última capa de la red neuronal está contenida $\hat{y}^{(i)}$, que denota la predicción para el i-ésimo ejemplo de entrenamiento.

Ilustración xiii. Red neuronal de 3 capas con vector de entrada y predicción



También, se puede observar que como se ha dicho la red propuesta está formada por 4 capas de nodos, pero siendo solo las 2 internas lo que se conoce como capas ocultas.

Como se tratan de forma similar a las capas exteriores (de entrada y de salida), y definiendo la variable $\boldsymbol{a}$ como un vector que agrupa todos los coeficientes de las capas mediante un

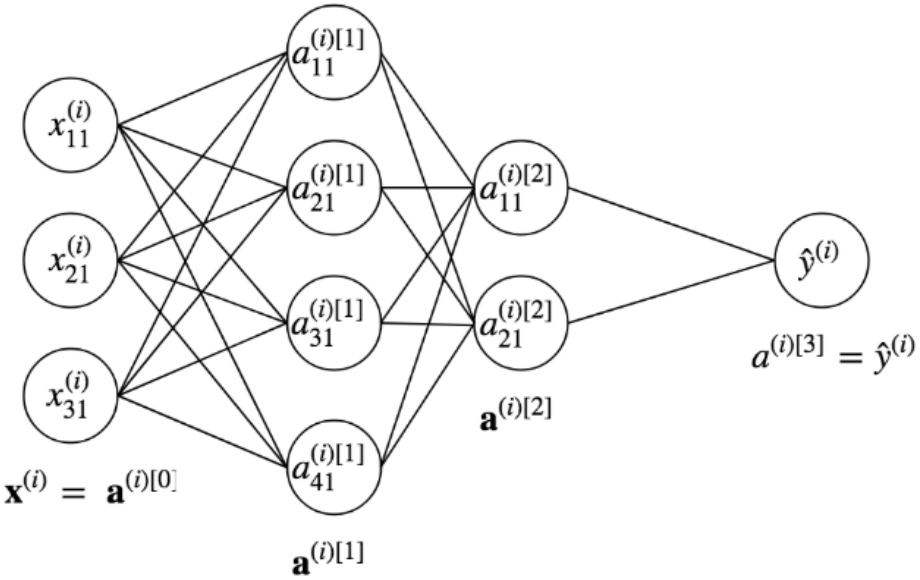
superíndice, en este caso *[j]* que indica el número de capa con el que se corresponde. Por tanto, técnicamente se puede afirmar que:

$$x^{(i)} = a^{(i)[0]} \quad \mathbf{y} \quad \hat{y}^{(i)} = a^{(i)[3]} \qquad \textbf{(23)}$$

Es decir, la primera columna de la matriz $a^{(i)[j]}$ se corresponde también con el vector de entrada y por el mismo modo, la última columna de esta se corresponderá con el vector de predicción.

Una vez conocido esto, la red se puede describir de la siguiente forma:

Ilustración xiv. Red neuronal de 3 capas con todos los coeficientes de activación



Para dejarlo más compacto se puede vectorizar cada uno de los vectores que corresponden con los valores todos los ejemplos de entrenamiento en cada capa "*j*", de la siguiente forma:

$$A^{[j]} = \begin{bmatrix} \vdots & \vdots & \cdots & \vdots \\ a^{(1)[j]} & a^{(2)[j]} & \cdots & a^{(m)[j]} \\ \vdots & \vdots & \cdots & \vdots \end{bmatrix} \qquad \textbf{(24)}$$

Esta matriz $A^{[j]}$ posee unas dimensiones de $(n^{a^{[j]}}, m)$, donde $n^{a^{[j]}}$ se corresponde con el número de unidades ocultas o nodos para la capa j y m el número de ejemplos de entrenamiento. Por ejemplo, para una mejor compresión:

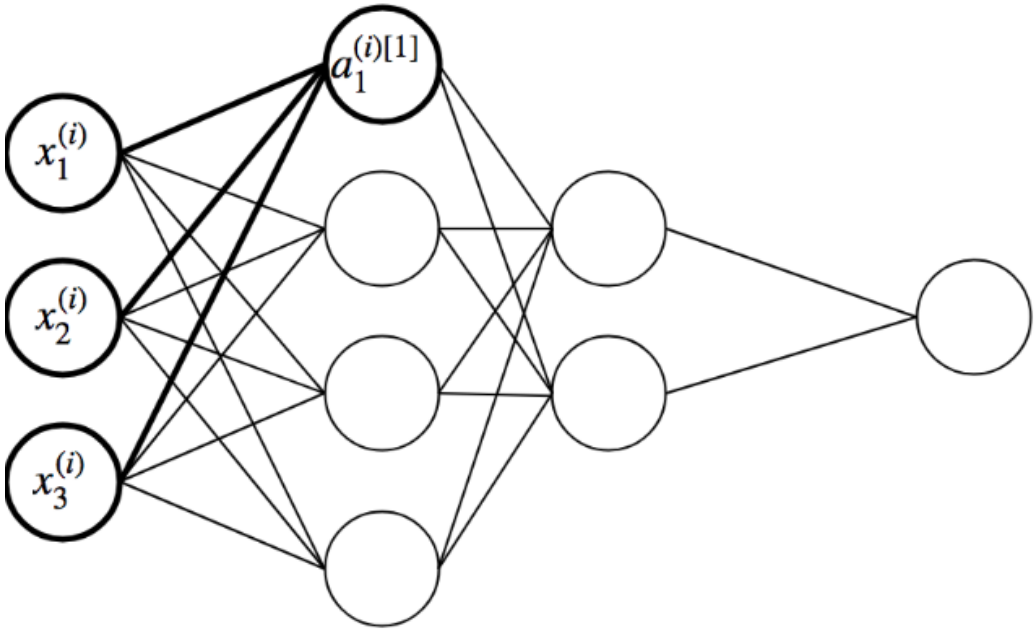
$$Si\ j\ =\ 1\,,\qquad n^{a^{[j]}}\ =\ 4\ \ entonces\ A^{[1]}\ será\ una\ matriz\ de\ dimensión\ (4, m)$$

Una vez aprendido un poco de nomenclatura y notación de este tipo de problemas se va a empezar a mostrar como realmente se calcula u obtiene cada uno de los valores de $a^{(i)[j]}$.

Cálculo de capa en capa

Primeramente, se empezará por el primer nodo de la primera capa oculta en su primer contacto de entrenamiento, es decir, ${a_1}^{(i)[1]}$.

Ilustración xv. Cálculo del primer nodo de la red



En el diagrama previo se observa como las entradas a este nodo al igual que a los demás están compuestas por combinaciones y ponderaciones de todos los términos del vector característico de entrada (para un determinado ejemplo de entrenamiento concreto). Por tanto, será el resultante de esta combinación el valor entrante en esta primera capa oculta.

Sabiendo esto, en objeto de calcular ${a_1}^{(i)[1]}$, se seleccionan cada uno de los términos del vector de entrada para cada numero de entrenamiento i-ésimo y se multiplica o ponderan por unos pesos, llamados comúnmente en la jerga del aprendizaje automático, “weights” (del inglés) que ya han sido citados con anterioridad.

Por lo que, se pueden expresar estos pesos también como una matriz con dimensiones asociadas a la cantidad de términos que tenga en vector de entrada, es decir su longitud (n_x), y al número de unidades neuronales que posea esa capa ($n^{a^{[j]}}$).

Por tanto, de forma generalizada, la matriz $W^{[j]}$ tendrá dimensiones ($n^{a^{[j]}}, n_x$).

En este caso, por ejemplo, se puede ver que las dimensiones para $W^{[1]}$ serán *(4,3).*

Expresando en su forma más general la matriz de pesos con respecto a cada capa:

$$W^{[j]} = \begin{bmatrix} {W_{11}}^{[j]} {W_{12}}^{[j]} \dots {W_{1\,n_x}}^{[j]} \\ {W_{21}}^{[j]} \quad {W_{22}}^{[j]} \dots {W_{2\,n_x}}^{[j]} \\ \vdots \quad \vdots \quad \vdots \quad \vdots \\ {W_{n^{a^{[j]}}1}}^{[j]} {W_{n^{a^{[j]}}2}}^{[j]} \dots {W_{n^{a^{[j]}}n_x}}^{[j]} \end{bmatrix} \qquad (25)$$

Vectorizando y simplificándola:

$$W^{[j]} = \begin{bmatrix} {W_1}^{[j]} \\ {W_2}^{[j]} \\ \vdots \\ {W_{n^{a^{[j]}}}}^{[j]} \end{bmatrix} \qquad (26)$$

siendo $$W_{n^{a^{[j]}}}{}^{[j]} = \begin{bmatrix} {W_{n^{a^{[j]}}1}}^{[j]} & {W_{n^{a^{[j]}}2}}^{[j]} & \cdots & {W_{n^{a^{[j]}}n_x}}^{[j]} \end{bmatrix} \qquad (27)$$

Una vez conocida la matriz de pesos, $W^{[j]}$ se puede obtener el valor resultante de estas combinaciones entre los parámetros característicos de entrada y sus ponderaciones, con lo que resulta una nueva variable, ${z_k}^{(i)[j]}$, siendo $k = 0,1,2 \dots n^{a^{[j]}}$.

Esta nueva variable de entrada a las neuronas se denota tanto por el nodo del que se trate y su capa como por el ejemplo de entrenamiento en el que se encuentre.

Su expresión analítica es de la forma:

$$z_k{}^{(i)[j]} = W_k{}^{[j]} \times a_k{}^{(i)[j]} + b_k{}^{[j]} \quad \textbf{(28)}$$

Como se puede ver se ha añadido también un término que se conoce del inglés como ***bias.*** Este es un parámetro, a parte de los pesos, que la red también va aprendiendo y modificando. ¿Pero de que se trata realmente?

Pues bien, no es más que término que se encarga de desplazar la no-linealidad de la función de activación que exista en los nodos de manera que el modelo sea capaz de aprender de forma más flexible, ya que, esta capacidad de no linealidad es la base para que la red resuelva cualquier tipo de problema.

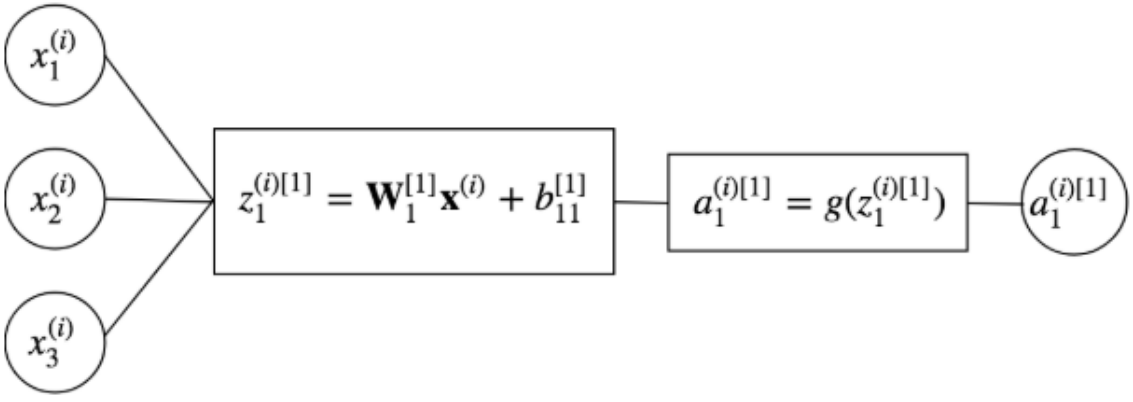
Por último, para asociar la salida de cada neurona, $a_k{}^{(i)[j]}$, a la entrada de esta justo previamente citada, $z_k{}^{(i)[j]}$, únicamente se necesita de la función de activación pertinente para su cálculo final.

Es decir, $$a_k{}^{(i)[j]} = g(z_k{}^{(i)[j]}), \quad \textbf{(29)}$$

donde $g()$ será la función de activación interna de cada neurona.

Una vez que ya se entiende de manera general como realizar el cálculo de los coeficientes que resultan de cada nodo y antes de pasar a explicar cómo funciona la propagación hacia adelante se detalla a continuación un breve ejemplo analítico de cómo se formaría la salida de la neurona del primer nodo en la primera capa oculta, $a_1{}^{(i)[1]}$, como se ha indicado con anterioridad.

Ilustración xvi. Esquema del cálculo analítico de la salida del primer nodo



Como se puede apreciar y como se ha explicado de forma general, partiendo del vector de entrada, ponderándolo con los pesos correspondientes y adhiriéndole el bias se obtiene lo que será la entrada al nodo 1 de la primera capa oculta (j = 1).

Una vez obtenido esto, se pasa esta variable por la activación de este nodo (normalmente al tratarse de la primera capa oculta de una red profunda con varias capas, la activación más común se realizará con la función ReLU), y se llega hasta, ${a_1}^{(i)[1]}$ que pasa de ser la salida de la neurona de la capa con j = 1 a la entrada de otra u otras neuronas de la siguiente capa de la red.

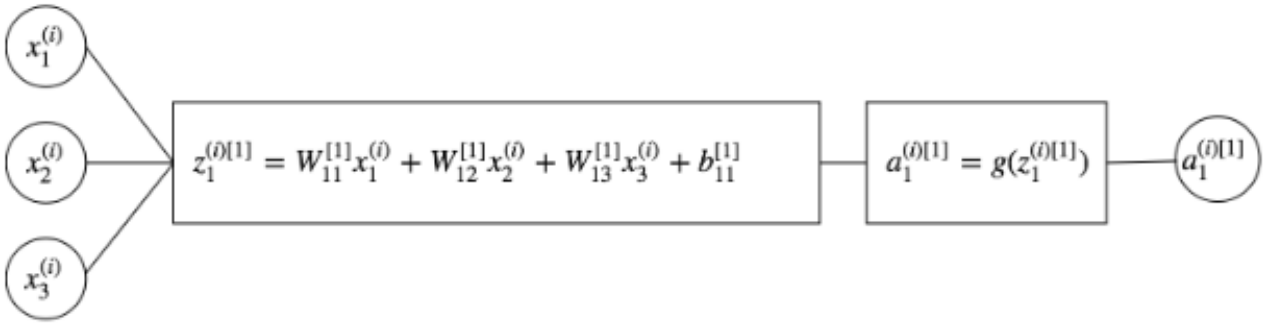
Para detallar la combinación de los pesos con la entrada de la red un poco más en profundidad, como se ha visto que:

$$W_1{}^{[1]} = \begin{bmatrix} W_{11}{}^{[1]} & W_{12}{}^{[1]} & W_{13}{}^{[1]} \end{bmatrix} \quad (30)$$

Y también, $$x^{(i)} = [{x_1}^{(i)} \quad {x_2}^{(i)} \quad {x_3}^{(i)}] \quad (31)$$

Entonces, el diagrama resultante se puede expresar más desglosado como:

Ilustración xvii. Esquema analítico del primer nodo desglosado



Una vez concluida esta breve explicación sobre como calcular los parámetros y activaciones de un nodo, mediante la propagación hacia adelante se conseguirá llegar al nodo de la capa de salida, del cual resulta la predicción dada por el sistema.

Por tanto, en el siguiente apartado se expone, apoyándose sobre la misma configuración de red como es este proceso de propagación.

Forward propagation

Este proceso básicamente consiste en llegar a obtener una salida predictiva, lo que ya se ha denotado como $\hat{y}^{(i)}$, partiendo de un conjunto de características de entrada agrupadas de manera ordenada en un vector, $x^{(i)}$.

Como se expuso anteriormente, se observa en la figura X como se forma la configuración de esta red:

$x^{(i)} = a^{(i)[0]}$*, formada por un vector característico de (*3×1*)*

$a^{(i)[1]}$*, formada por (*4×1*) activaciones en la primera capa oculta*

$a^{(i)[2]}$*, formada por (*2×1*) activaciones en la segunda capa oculta*

$a^{(i)[3]} = \hat{y}^{(i)}$*, formada por el valor de salida (*1×1*), donde* $\hat{y}^{(i)}$*es la probabilidad de que* $y^{(i)}$ *sea la clase positiva si se habla de clasificación binaria*

Aunque perfectamente podría explicarse con una red más general, esta puede facilitar bastante el entendimiento del concepto.

Por tanto, para resumir, el proceso de propagación hacia adelante será de la forma:

$$x^{(i)} \rightarrow a^{(i)[1]} \rightarrow a^{(i)[2]} \rightarrow \hat{y}^{(i)}$$

A continuación, se va a detallar los pasos internos de cada una de estas transiciones.

DESDE LA ENTRADA DE ENTRENAMIENTO HASTA 1ª CAPA OCULTA ($x^{(i)} \rightarrow a^{(i)[1]}$)

El primer paso que ocurre cuando se hace transición desde el vector de entrada al entrenamiento en primera instancia, es decir en el primer ejemplo i de entrenamiento, hasta las activaciones de la primera capa oculta de la red será comenzar multiplicando estas características de entrada por los parámetros internos de las neuronas de esta capa (los "weights" y el "bias"), ya previamente detallados.

De esta forma, se obtendrá la entrada resultante a la fase de activación en esta primera capa, $z^{(i)[1]}$. Para ampliar un poco más se hace un breve análisis de dimensionamiento:

$$z^{(i)[1]} = W^{[1]} \times x^{(i)} + b^{[1]} \quad (32)$$

Como se ha presentado anteriormente, la dimensión de la matriz de pesos depende básicamente del número de neuronas que exista en una determinada capa y de la longitud del vector de entrada a esa capa. En este caso, como se ha visto, la primera capa oculta está constituida por 4 unidades neuronales y la entrada a esta capa que será el vector $x^{(i)}$ se ha establecido que tiene una longitud, $n_x = 3$.

Por lo tanto, se consigue una matriz $W^{[1]}$ de dimensión (4×3).

Por el otro lado, el vector característico de entrada a esta transición $x^{(i)}$ va a tener una dimensión de (3×1).

Fácilmente puede verse como realizando el producto entre ambas matrices de obtendrá un vector de dimensión (4×1), al que se le añade también el término del "bias", el cual sus dimensiones se adaptan a las resultantes del producto matricial, es decir, (4×1). Con todo esto se llega a $z^{(i)[1]}$, llamada comúnmente matriz de actividad, la cual será la encargada de atravesar las unidades de la capa aplicando estas la correspondiente activación a cada elemento de $z^{(i)[1]}$. En este caso se había escogido la función de activación ReLU, ya que se trata de una activación de capa interna.

Por tanto, siendo *g()* la función ReLU se obtendrá finalmente la salida de la capa determinada, lo que se llama la matriz de activación:

$$a^{(i)[1]} = g(z^{(i)[1]}) \quad (33)$$

De modo que se aplica la activación a la matriz de actividad con dimensión (4×1) elemento a elemento, por lo que, la dimensión de la matriz resultante $a^{(i)[1]}$ también va a tener este mismo dimensionamiento.

Una vez obtenida la matriz de activación de esta primera capa oculta, que será la entrada necesaria para la siguiente transición se pasa a la siguiente fase del entrenamiento.

DESDE LA 1ª CAPA OCULTA HASTA LA 2ª CAPA OCULTA ($a^{(i)[1]} \rightarrow a^{(i)[2]}$)

Esta sección va a ser prácticamente similar a la sección previa, solo que la entradas y salidas resultantes no serán las mismas, aunque si lo será el procedimiento.

Se empieza multiplicando y ponderando el vector de entrada a esta nueva capa con los parámetros asociados a esta (que nunca tendrán por qué coincidir con los de otra capa).

En este caso la entrada se corresponde con la resultante de la primera transición, es decir, $a^{(i)[1]}$ que como se ha explicado tiene dimensiones (4×1).

Expresando la matriz de actividad de esta segunda capa interna de manera analítica:

$$z^{(i)[2]} = W^{[2]} \times a^{(i)[1]} + b^{[2]} \quad (34)$$

Donde $W^{[2]}$ es una matriz de (2×4), ya que se tiene dos nodos en esta capa y la longitud de la entrada a la capa es igual a 4.

Obviando un poco por encima lo explicado anteriormente, se obtiene una matriz de actividad con dimensión (2×1).

Una vez que se ha conseguido calcular $z^{(i)[2]}$, se vuelve a aplicar la activación de esta capa que también se corresponderá con la función ReLU de manera que se concluye esta transición con la obtención de la matriz saliente de activación:

$$a^{(i)[2]} = g(z^{(i)[2]}) \quad (35)$$

Siendo $a^{(i)[2]}$ de las mismas dimensiones que $z^{(i)[2]}$, es decir, (2×1).

Obtenido esto, se concluye este proceso explicando la última fase.

DESDE LA 2ª CAPA OCULTA HASTA LA CAPA DE SALIDA (Predicción) ($\boldsymbol{a^{(i)[1]} \rightarrow a^{(i)[2]}}$)

En esta sección se comienza de la misma forma que en las anteriores, es decir, obteniendo la matriz de actividad a partir de la matriz de entrada a esta capa y los parámetros internos de la misma:

$$z^{(i)[3]} = W^{[3]} \times a^{(i)[2]} + b^{[3]} \quad (36)$$

Siendo $W^{[3]}$ una matriz con dimensiones (1×2), $a^{(i)[2]}$ con dimensión (2×1) y, por lo tanto, $z^{(i)[3]}$ y $b^{[3]}$ serán simplemente matrices de (1×1).

En este caso, la función de activación que se le aplique a la matriz de actividad no va a ser la función ReLU si no que se usará la función sigmoide, $\sigma(z)$, que ya se ha comentado en apartados anteriores.

Esta activación suele ser usada como ya se explicó en la última capa normalmente. Esta razón se debe a que la función sigmoide posee una gran capacidad para aplastar cualquier valor y tomar otro entre el intervalo [0 , 1], de modo que resulta una cualidad muy útil para problemas de clasificación con predicción, ya que, la salida de la activación será finalmente un valor entre este intervalo y nos servirá de valor predictivo.

Por tanto, se tiene que:

$$\hat{y}^{(i)} = a^{(i)[3]} = \sigma(z^{(i)[3]}) \quad (37)$$

Donde $\hat{y}^{(i)}$ será un valor concreto de predicción en cada uno de los ejemplos de entrenamiento.

Aquí finaliza la primera etapa del proceso propagativo hacia adelante, es decir, en el primer contacto de entrenamiento *(i = 1*). Cada vez que la red vuelva a empezar a entrenarse y actualice todos estos parámetros explicados se aumentará el número de entrenamientos realizados y por consiguiente todas las matrices i-ésimas irán aumentando su dimensionamiento hasta llegar a *i = m.*

Este algoritmo complementario de realimentación y actualización de los parámetros es conocido como proceso de propagación hacia atrás y será expuesto detalladamente a continuación.

Conocimiento previo al proceso de propagación hacia atrás

El objetivo de la retropropagación es bastante claro, es necesario calcular las derivadas parciales de nuestros parámetros con respecto a la función de costo (***J***) para usarlo en el descenso de gradiente. La parte difícil radica en hacer un seguimiento de los cálculos, ya que cada derivada parcial de los parámetros en cada capa se basa en las entradas de la capa anterior. Tal vez también es el hecho de que se esté yendo hacia atrás hace que este proceso sea más difícil para el entendimiento.

La propagación hacia atrás puede ser usada de diversas formas, pero para este caso, como se viene exponiendo se utilizará básicamente para el entrenamiento de un clasificador binario.

Para ello, al igual que en el proceso anteriormente descrito será de utilidad una red con una arquitectura neuronal de 3 capas, cada una con su respectivo número de unidades.

Este proceso como se acaba de decir se basa principalmente en la función de pérdidas de costo de la red, por lo que, se comenzará por la comprensión de la misma.

La notación matemática, de los parámetros, variables, activaciones y demás términos necesarios, usada en este apartado será la misma que en el anterior.

Una vez dicho esto se procede a presentar la función de pérdidas.

FUNCION DE PERDIDAS

En el apartado previo, se explicó cómo se podía avanzar desde la entrada de entrenamiento $x^{(i)}$, hasta la predicción del sistema $\hat{y}^{(i)}$.

Por tanto, ¿cómo se puede saber cómo de bien realiza el modelo esta predicción?

Pues bien, la cuestión clave al fin y al cabo cuando el modelo se entrena será observar cuanto de cerca se encuentra la predicción del sistema con respecto a la salida real asociada a la entrada de la red, $y^{(i)}$ que se corresponde con las clases del clasificador binario [0/1].

Con el objeto de medir el error existente entre estos dos valores, aparece la llamada función de pérdidas.

Esta función puede ser aproximada siguiendo varios métodos matemáticos diferentes. El más básico posiblemente sea el calculado con el error cuadrático medio entre ambas variables, siendo su expresión:

$$\mathcal{L}(\hat{y}, y) = \frac{1}{2}(\hat{y} - y)^2 \quad (38)$$

Esta es una función bastante simple, que lo único que hace es calcular la diferencia entre la salida real y la predicción del sistema, elevarlo al cuadrado y hacer la media del valor cuadrático para que la función sea positiva en todo su dominio.

Resulta que este tipo de método no es el más apropiado en problemas de aprendizaje automático con regresión logística, ya que, cuando se intenta entrenar y que los parámetros aprendan, el problema de optimización no es convexo.

Por tanto, se va a elegir otro tipo de función de pérdidas para este tipo de tecnologías, la llamada función de pérdidas de entropía cruzada.

Esta nueva función puede ser expresada de la siguiente forma:

$$\mathcal{L}(\hat{y}, y) = -y \cdot \log(\hat{y}) - (1 - y) \cdot \log(1 - \hat{y}) \quad (39)$$

Sabiendo que el objetivo de la optimización del problema será conseguir minimizar el valor de la función de pérdidas, se puede observar fácilmente en la expresión que:

- Si el valor real de la salida, es decir, $y = 1$, sustituyendo en la ecuación:

$$\mathcal{L}(\hat{y}, y = 1) = -\log(\hat{y}) \quad (40)$$

Por lo que, para obtener el mínimo valor de pérdida habrá que conseguir una predicción $\hat{y}$ con valores cercanos a su máximo, ya que, esta variable posee su rango de valores entre (0,1) gracias a la función de activación sigmoide que ya se ha explicado.

Inversamente:

- Si el valor de $y = 0$, sustituyendo en la expresión general:

$$\mathcal{L}(\hat{y}, y = 0) = -\log(1 - \hat{y}) \quad (41)$$

Donde se necesita que la predicción del sistema obtenga valores lo más cercano posible a 0, ya que, será en ese punto donde se minimice la función de pérdidas $\mathcal{L}(\hat{y}, y)$.

Por tanto, se ha mostrado como funciona esta función, pero, ¿de dónde sale realmente?

Si se toma y como una variable aleatoria de Bernoulli. Sabiendo que la función de probabilidad de masa para una variable aleatoria de Bernoulli se expresa de forma general como:

$$p(k|p) = k^p(1-k)^{(1-p)} \quad (42)$$

Donde p es la probabilidad de que k sea igual a un valor concreto.

Si se sustituye k por nuestro valor real a obtener y , siendo $\hat{y}$ la probabilidad de que la predicción sea lo más parecida al valor real, se tiene que:

$$p(y|\hat{y}\,, x) = y^{\hat{y}}\,(1-y)^{(1-\hat{y}\,)} \quad (43)$$

Pero, ¿por qué no se trabaja directamente con esta función? La razón es porque en el proceso que se explicará a continuación de propagación hacia atrás se necesita de la derivada de la función de pérdidas y trabajando con esta función se puede hacer bastante enrevesado.

Para ello, se realizan una serie de cambios en la expresión, que no alteran el objetivo y hace que la función se incremente de manera suave y continua, lo que facilita el cálculo de su derivada.

Esta modificación se lleva a cabo realizando el logaritmo en ambas partes de la expresión:

$$log(p(y|\hat{y}\,, x)) = log(y^{\hat{y}}\,(1-y)^{(1-\hat{y}\,)}) \quad (44)$$

$$log(p(y|\hat{y}\,, x)) = y \cdot \log(\hat{y}) + (1-y) \cdot \log(1-\hat{y}) \quad (45)$$

Por tanto, se llega a obtener que:

$$\mathcal{L}(\hat{y}, y) \;=\; -log(p(y|\hat{y}\,, x)) \quad (46)$$

De esta expresión, se puede decir que cuando se quiere minimizar la función de pérdidas $\mathcal{L}(\hat{y}, y)$, lo que realmente significa es que se necesita maximizar la probabilidad de que el valor real de salida se corresponda con su valor máximo estipulado, dada la predicción $\hat{y}$ y el vector de entrada característico x.

De este modo, se representa la función de pérdidas para únicamente un ejemplo de entrenamiento i.

Tomando la función de costo $\mathcal{J}$, como la media de la función de pérdidas en cada uno de los entrenamientos que se realicen en la red, siendo m el número de entrenamientos total se obtiene que:

$$\mathcal{J} = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}\left(\hat{y}^{(i)}, y^{(i)}\right) \quad (47)$$

Dependiendo de si m es igual al número de entrenamientos o no, el algoritmo de optimización se nombrará de una forma u otra. Por ejemplo:

- Si m si corresponde con el número de entrenamientos total realizados el proceso se conoce como descenso de gradiente por lote (más conocido en inglés como "batch gradient descent").
- Si m es menor que el total de estos, este algoritmo se conoce como descenso de gradiente por mini lote (en inglés, "mini-batch gradient descent").

Una vez comprendido el concepto de función de costo y, por tanto, de pérdidas se va a explicar cómo se realiza el cálculo necesario de los gradientes de esta función para cada uno de los entrenamientos, el cual será el principal objetivo para obtener el proceso completo de propagación hacia atrás.

Propagación hacia atrás

Introducción

Este proceso se realiza con el principal fin de calcular los gradientes de la función de coste en cada uno de los ejemplos de entrenamiento con respecto a cada uno de los parámetros útiles de la red de manera que se actualicen y modifiquen si existiese mejoría en el resultado del coste de la red, es decir si se minimiza ese coste a medida que avanzan los entrenamientos.

El gradiente es simplemente la derivada parcial de una función con respecto a cada uno de los parámetros de entrada.

A modo de ejemplo, el gradiente de la función de costo $\mathcal{J}$, con respecto al peso que conecta el tercer elemento del vector de entrada con el segundo nodo de la primera capa oculta sería:

$$\frac{\partial \mathcal{J}}{\partial w_{23}^{[1]}} \quad (48)$$

Conocido esto, lo que se hace en la retropropagación es obtener el valor de cada uno de los gradientes de todos los parámetros de la red, almacenarlos de una manera determinada y actualizarlos si fuese necesario.

Por ejemplo, recordando la arquitectura de la red propuesta en la propagación hacia adelante, mostrada en la *ilustración xiv*, se puede observar como sumando cada una de las dimensiones de las matrices de pesos y de bias en cada capa:

$$W^{[1]}: (4 \times 3) \rightarrow 12 \ / \ b^{[1]}: (4 \times 1) \rightarrow 4$$

$$W^{[2]}: (2 \times 4) \rightarrow 8 \ / \ b^{[2]}: (2 \times 1) \rightarrow 2$$

$$W^{[3]}: (1 \times 2) \rightarrow 2 \ / \ b^{[3]}: (1 \times 1) \rightarrow 1$$

Se obtiene un total de 29 parámetros diferentes con los que entrenar y aprender a resolver el problema.

De igual manera que se hizo anteriormente, estos parámetros pueden ser agrupados y vectorizados en matrices para cada una de las capas ayudando a que la notación del problema resulte más sencilla.

Por ejemplo, se puede representar el gradiente de la función de coste $\mathcal{J}$, con respecto a la matriz $W^{[2]}$, siendo esta la matriz de los pesos correspondientes a todos los nodos de la segunda capa oculta.

De esto resulta:

$$\boldsymbol{d}W^{[2]} = \begin{bmatrix} \frac{\partial \mathcal{J}}{\partial w_{11}^{[2]}} & \frac{\partial \mathcal{J}}{\partial w_{12}^{[2]}} & \frac{\partial \mathcal{J}}{\partial w_{13}^{[2]}} & \frac{\partial \mathcal{J}}{\partial w_{14}^{[2]}} \\ \frac{\partial \mathcal{J}}{\partial w_{21}^{[2]}} & \frac{\partial \mathcal{J}}{\partial w_{22}^{[2]}} & \frac{\partial \mathcal{J}}{\partial w_{23}^{[2]}} & \frac{\partial \mathcal{J}}{\partial w_{24}^{[2]}} \end{bmatrix} \qquad \textbf{(49)}$$

Se observa a simple vista que las dimensiones de la matriz $dW^{[2]}$ se corresponden con las de $W^{[2]}$, lo que hace que la actualización del gradiente una operación elemento a elemento bastante asequible.

Por tanto, también se puede expresar de esta forma:

$$\frac{\partial \mathcal{J}}{\partial W^{[2]}} = \boldsymbol{d}W^{[2]} \qquad \textbf{(50)}$$

Habiendo entendido este concepto de notación se obtienen por tanto 6 matrices diferentes que forman los 29 parámetros de la red:

$$\left(\boldsymbol{d}W^{[1]}, \boldsymbol{d}b^{[1]}, \boldsymbol{d}W^{[2]}, \boldsymbol{d}b^{[2]}, \boldsymbol{d}W^{[3]}, \boldsymbol{d}b^{[3]}\right) \quad (51)$$

Hay que tener en cuenta como ya se ha dicho, que cada una de estas matrices poseen la misma dimensión que su matriz de pesos o de bias asociada.

Implementación de la retro propagación

En la propagación hacia adelante, el objetivo principal era llegar obtener una salida de predicción $\hat{y}^{(i)}$, a partir de una serie de parámetros y un vector de entrada para cada uno de los ejemplos entrenamiento.

Ahora en la propagación hacia atrás, por ejemplo, en este caso que se ha propuesto, el objetivo será calcular las 6 matrices de gradiente.

Este proceso se realiza del final de la red hacia el principio, por lo tanto, se comenzará calculando primero las matrices:

$$\boldsymbol{d}W^{[3]}, \boldsymbol{d}b^{[3]} \quad (52)$$

Para ello, hay que hacerse una cuestión antes de comenzar. ¿Cuál es la ecuación que relaciona la función de coste con las matrices de pesos y bias?

Pues bien, para generalizar un poco más se va a denotar la predicción $\hat{y}^{(i)}$, como $a^{[3]}$. Teniendo esto se expresa la función de coste con respecto a los parámetros de la capa correspondiente de la forma:

$$\frac{\partial \mathcal{J}}{\partial W^{[3]}} = \frac{d\mathcal{J}}{da^{[3]}} \frac{da^{[3]}}{dz^{[3]}} \frac{\partial z^{[3]}}{\partial W^{[3]}} \quad (53)$$

$$\frac{\partial \mathcal{J}}{\partial b^{[3]}} = \frac{d\mathcal{J}}{da^{[3]}} \frac{da^{[3]}}{dz^{[3]}} \frac{\partial z^{[3]}}{\partial b^{[3]}} \quad (54)$$

Antes de seguir, se puede observar que la primera ecuación calcula la derivada parcial de $\mathcal{J}$, con respecto a $W^{[3]}$, que se trata de un vector con dimensiones (1,2). De la misma manera, puede referirse a la segunda ecuación, pero, con respecto a $b^{[3]}$.

También, se debe notar que en algunos casos la operación de derivar se marca con "d" y en otros casos con "∂". La razón es porque el primer operador corresponde con derivadas que

se refieren a una sola variable, por el contrario, el segundo operador de gradiente se asocia con operaciones de más de una variable, es decir, derivadas parciales.

Teniendo en cuenta la arquitectura básica propuesta se va a mostrar los pasos a seguir para calcular estos parámetros. Se ha hecho la demostración del cálculo con esta configuración para facilitar la comprensión, pero obviamente, si se entiende bien el concepto, este proceso se debería de poder escalar a cualquier tipo de configuración de red.

Por tanto, se va a comenzar calculando la derivada de $\mathcal{J}$, con respecto a $a^{[3]}$:

$$\mathcal{J}\left(a^{[3]}, y\right) = -y \cdot \log\left(a^{[3]}\right) - (1-y) \cdot \log\left(1-a^{[3]}\right) \quad (55)$$

Si se realiza la derivada a ambos lados de la expresión y se simplifica, se obtiene:

$$\frac{d\mathcal{J}}{da^{[3]}} = \frac{-y}{a^{[3]}} + \frac{(1-y)}{1-a^{[3]}} \quad (56)$$

Ahora se hace lo mismo, pero, con la derivada de $a^{[3]}$ con respecto a $z^{[3]}$:

$a^{[3]} = \sigma\left(z^{[3]}\right)$ siendo $\sigma()$, la función de activación sigmoide explicada previamente.

Entonces, $$a^{[3]} = \frac{1}{1+e^{-z^{[3]}}} \quad (57)$$

Derivando al igual que en el paso anterior, pero con respecto a $z^{[3]}$ y simplificando la expresión resultante, se tiene que:

$$\frac{da^{[3]}}{dz^{[3]}} = \frac{1}{1+e^{-z^{[3]}}} \frac{1}{1+e^{-z^{[3]}}} e^{-z^{[3]}} \quad (58)$$

Notando que:

$$e^{-z^{[3]}} = \frac{1-a^{[3]}}{a^{[3]}} \quad (59)$$

Volviendo a la derivada, sustituyendo lo anterior y simplificando se obtiene que:

$$\frac{da^{[3]}}{dz^{[3]}} = a^{[3]}\,(1 - a^{[3]}) \qquad (60)$$

Finalmente, para obtener las dos primeras matrices de la primera capa empezando por atrás habrá que calcular las derivadas parciales de $z^{[3]}$, con respecto a $W^{[3]}$ y $b^{[3]}$.

Conociendo que $W^{[3]}$ es un vector formado por:

$$\mathrm{W}^{[3]} = \begin{bmatrix} \mathrm{W}_{11}^{[3]} & \mathrm{W}_{12}^{[3]} \end{bmatrix} \qquad (61)$$

Se puede descomponer:

$$z^{[3]} = W^{[3]} a^{[2]} + b^{[3]} \qquad (62)$$

En:

$$z^{[3]} = \begin{bmatrix} W_{11}^{[3]} & W_{12}^{[3]} \end{bmatrix} \begin{bmatrix} a_{11}^{[2]} \\ a_{21}^{[2]} \end{bmatrix} + b^{[3]} \qquad (63)$$

$$z^{[3]} = W_{11}^{[3]}\, a_{11}^{[2]} + W_{12}^{[3]}\, a_{21}^{[2]} + b^{[3]} \qquad (64)$$

Teniendo esto, se calculan sus derivadas parciales:

$$\frac{\partial z^{[3]}}{\partial w_{11}^{[3]}} = a_{11}^{[2]} \qquad (65)$$

$$\frac{\partial z^{[3]}}{\partial w_{12}^{[3]}} = a_{21}^{[2]} \qquad (66)$$

Por tanto, generalizando:

$$\frac{\partial z^{[3]}}{\partial W^{[3]}} = \begin{bmatrix} a_{11}^{[2]} & a_{21}^{[2]} \end{bmatrix} \qquad (67)$$

Interesantemente, se puede apreciar como la derivada parcial tendrá dimensiones de (1,2) mientras el vector de entrada a la capa 3 posee dimensiones de (2,1).

Por tanto, se puede decir simplemente que la derivada parcial de $z^{[3]}$ con respecto a $W^{[3]}$ es únicamente la matriz de activación de la capa anterior, es decir la segunda, pero traspuesta:

$$\frac{\partial z^{[3]}}{\partial W^{[3]}} = a^{[2]T} \quad (68)$$

Realizando las mismas operaciones, pero en lugar de con la matriz de pesos usando la de bias se tiene que:

$$\frac{\partial z^{[3]}}{\partial b^{[3]}} = 1 \quad (69)$$

Una vez obtenido esto, si se conecta $\frac{d\mathcal{J}}{da^{[3]}}$ y $\frac{da^{[3]}}{dz^{[3]}}$ para simplificar y formar $\frac{d\mathcal{J}}{dz^{[3]}}$:

$$\frac{d\mathcal{J}}{dz^{[3]}} = \frac{d\mathcal{J}}{da^{[3]}} \frac{da^{[3]}}{dz^{[3]}} \quad (70)$$

$$= \left(\frac{-y}{a^{[3]}} + \frac{(1-y)}{1-a^{[3]}}\right) a^{[3]} (1 - a^{[3]}) \quad (71)$$

Operando un poco se consigue:

$$\frac{d\mathcal{J}}{dz^{[3]}} = a^{[3]} - y = \delta^{[3]} \quad (72)$$

Sustituyendo en la ecuación de las derivadas principales a calcular:

$$\frac{\partial \mathcal{J}}{\partial W^{[3]}} = (a^{[3]} - y)a^{[2]T} = \delta^{[3]} a^{[2]T} \quad (73)$$

$$\frac{\partial \mathcal{J}}{\partial b^{[3]}} = \frac{d\mathcal{J}}{dz^{[3]}} = a^{[3]} - y = \delta^{[3]} \quad (74)$$

Ya se conocen las dos primeras matrices correspondientes a la capa 3, ahora habrá que calcular las 2 de la capa 2 y finalmente las 2 de la primera capa oculta.

Avanzando, por tanto, a la segunda capa, las siguientes derivadas matriciales a calcular serán:

$$\frac{\partial \mathcal{J}}{\partial W^{[2]}} = \frac{d\mathcal{J}}{dz^{[3]}} \frac{dz^{[3]}}{da^{[2]}} \frac{da^{[2]}}{dz^{[2]}} \frac{\partial z^{[2]}}{\partial W^{[2]}} \quad (75)$$

$$\frac{\partial \mathcal{J}}{\partial b^{[2]}} = \frac{d\mathcal{J}}{dz^{[3]}} \frac{dz^{[3]}}{da^{[2]}} \frac{da^{[2]}}{dz^{[2]}} \frac{\partial z^{[2]}}{\partial b^{[2]}} \quad (76)$$

Si se conoce que:

$$\frac{d\mathcal{J}}{dz^{[3]}} = \frac{d\mathcal{J}}{da^{[3]}} \frac{da^{[3]}}{dz^{[3]}} \quad (77)$$

Y del cálculo de las ecuaciones de la capa anterior ya se conoce esta expresión, no tendrá que volver a calcularse:

$$\frac{d\mathcal{J}}{dz^{[3]}} = a^{[3]} - y \quad (78)$$

Operando entonces, de igual manera que para el cálculo de las matrices anteriores se obtiene:

$$\frac{\partial \mathcal{J}}{\partial W^{[2]}} = \delta^{[2]} a^{[1]T} \quad (79)$$

$$\frac{\partial \mathcal{J}}{\partial b^{[2]}} = \delta^{[2]} \quad (80)$$

Siendo:

$$\delta^{[2]} = W^{[3]T} \delta^{[3]} * g'\left(z^{[2]}\right) \quad (81)$$

donde $g'()$ se corresponde con la derivada de la función de activación ReLU.

De la misma forma, las últimas dos expresiones a calcular correspondientes con la primera de las capas ocultas serán:

$$\frac{\partial \mathcal{J}}{\partial W^{[1]}} = \frac{d\mathcal{J}}{dz^{[2]}} \frac{dz^{[2]}}{da^{[1]}} \frac{da^{[1]}}{dz^{[1]}} \frac{\partial z^{[1]}}{\partial W^{[1]}} \quad (82)$$

$$\frac{\partial \mathcal{J}}{\partial b^{[1]}} = \frac{d\mathcal{J}}{dz^{[2]}} \frac{dz^{[2]}}{da^{[1]}} \frac{da^{[1]}}{dz^{[1]}} \frac{\partial z^{[1]}}{\partial b^{[1]}} \quad \textbf{(83)}$$

Siguiendo los mismos pasos y operaciones que anteriormente se puede llegar a obtener:

$$\frac{\partial \mathcal{J}}{\partial W^{[1]}} = \delta^{[1]} a^{[0]T} = \delta^{[1]} x^T \quad \textbf{(84)}$$

$$\frac{\partial \mathcal{J}}{\partial b^{[1]}} = \delta^{[1]} \quad \textbf{(85)}$$

$$\delta^{[1]} = W^{[2]T} \delta^{[2]} * g'\left(z^{[1]}\right) \quad \textbf{(86)}$$

Actualización de los parámetros

Una vez que se conoce cómo obtener de manera algebraica las matrices de pesos y sus respectivas matrices de gradiente, se procede a explicar de que forma se deben actualizar los parámetros de la red para que la función de error o coste descienda hasta alcanzar su mínimo y de esta manera conseguir el mejor resultado posible.

Esta actualización se realiza con la siguiente expresión:

$$W^{[l]} = W^{[l]} - \alpha \cdot \frac{\partial \mathcal{J}}{\partial W^{[l]}} \quad \textbf{(87)}$$

De la misma forma para actualizar las matrices de bias:

$$b^{[l]} = b^{[l]} - \alpha \cdot \frac{\partial \mathcal{J}}{\partial b^{[l]}} \quad \textbf{(88)}$$

Donde l se corresponde con la capa que se quiera actualizar y α se trata de la conocida tasa de aprendizaje (en inglés, “learning rate”).

Antes de seguir con el algoritmo de actualización se va a detallar brevemente en que consiste la configuración de la tasa de aprendizaje.

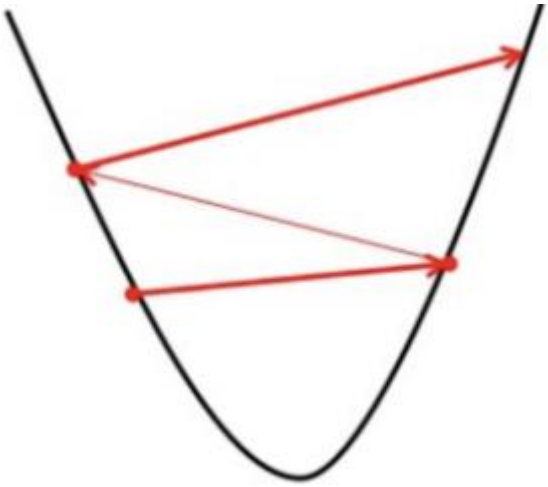
La tasa de aprendizaje es posiblemente el hiperparámetro más conocido en el mundo del aprendizaje automático. Esta se va modificando y adecuando por el programador de la red de modo que se obtengan óptimos resultados de aprendizaje.

Por ejemplo, si la tasa de aprendizaje es demasiado pequeña, la red aprenderá de manera muy lenta y por tanto será un algoritmo pesado y poco eficiente.

Ilustración xviii. Función de aprendizaje con Learning Rate pequeño

Por otro lado, si la tasa de aprendizaje es demasiado grande, cuando los pesos sean actualizados será muy difícil conseguir el mínimo necesario, ya que, se irán obteniendo valores casi aleatorios de la curva de pesos, eso sí, saltándose siempre el mínimo.

Ilustración xix. Función de aprendizaje con Learning rate grande



Para llegar a obtener el valor óptimo, si se calcula manualmente se tendrá que inicializar un valor medio concreto, por ejemplo 0.01 e ir cambio dicho valor a medida que se observa cómo se comporta la red en su aprendizaje.

En la práctica realmente existen ya métodos de optimización estandarizados por defecto que modelan este hiperparámetro de manera que se obtiene un mejor resultado.

Para este proyecto en concreto se usará el algoritmo de optimización ADAM.

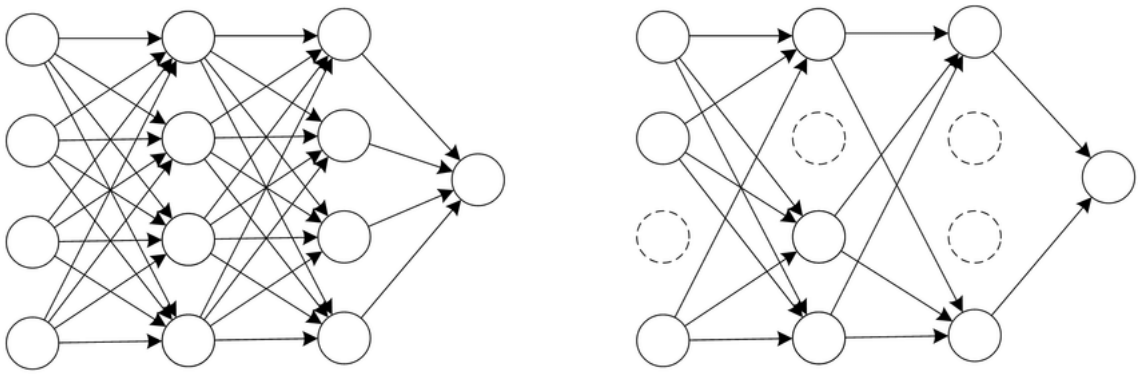
Por otro lado, también hay que tener en cuenta en este tipo de redes profundas que no se produzca un sobre entrenamiento de los datos introducidos, ya que, esto puede provocar que el aprendizaje converja y no se mejoren los resultados a medida que se propagan y se retro propagan estos datos. Para ello, existen mecanismos de regularización que mejoran este aspecto aceptablemente, haciendo uso de una técnica conocida como "**Dropout**".

Regularización con Dropout

Dropout es una técnica de regularización para modelos de redes neuronales propuesta por Srivastava y al. En su "paper" sobre Dropout publicado en 2014: "A Simple Way to Prevent Neural Networks from Overfitting" (Una manera simple de prevenir sobre entrenamiento en redes neuronales).

Dropout se trata de una técnica, en la cual, neuronas seleccionadas aleatoriamente van siendo ignoradas durante el entrenamiento. Estas son suprimidas ("dropped-out", de ahí su nombre) al azar, lo que significa que cualquier contribución de activación que pudieran efectuar dichas neuronas en el proceso de propagación hacia adelante queda neutralizada, no aplicándose, por consiguiente, ninguna actualización en los pesos de estas neuronas en la propagación hacia atrás.

Ilustración xx. Esquema del proceso de Dropout



(a) Standard Neural Network

(b) Network after Dropout

A medida que una red neuronal aprende, los pesos de las neuronas se asientan en su contexto dentro de la red. Los pesos de las neuronas están ajustados para características específicas que proporcionan una especialización concreta.

Debido a todo esto, serán las neuronas vecinas o cercanas a las abandonadas las que tendrán que intervenir y manejar la representación y predicción del sistema, sin tener en cuenta las neuronas que faltan y sin que afecte al proceso de aprendizaje.

El sistema, por lo tanto, también aprende las configuraciones neuronales que se van sucediendo aleatoriamente, incluyendo los pesos de cada una de ellas. El efecto de esto consiste en que la red se vuelve menos sensible a los pesos específicos de las neuronas. Esto a su vez da como resultado una red que es capaz de una mejor generalización del problema y es menos probable que se ajusten en exceso los datos de entrenamiento.

Esta técnica en programas de aprendizaje automático y profundo se implementa comúnmente gracias a un módulo específico que posee la API de Keras, detallada más adelante. Es implementada agregando una capa previa a cada capa oculta del modelo, teniendo en cuenta un parámetro que ajusta cuanto se quiere aplicar esta herramienta de regulación, yendo desde 0 (no se aplica "Dropout"), hasta 1 (se neutralizan todas las neuronas por lo que no existe aprendizaje).

En el caso de este proyecto se ha escogido, a base de prueba y error, un nivel de regulación "Dropout" igual a 0.3 para todas sus capas internas.

Una vez detallado esto, como se ha citado previamente se actualizan los pesos con los valores pertinentes en cada caso obteniéndose nuevos valores para los pesos, el siguiente paso del entrenamiento es volver a empezar con la propagación hacia adelante, el cálculo de la nueva función de coste, la nueva retropropagación y volver a actualizar todos los pesos.

Este proceso de descenso de gradiente se realiza una y otra vez de manera que el coste de la red se va minimizando si se ha configurado bien el modelo hasta un número de ejemplos de entrenamiento concreto al que se había llamado "m".

Se alcanza este número de entrenamiento cuando la función de coste se minimiza tanto hasta el punto que se converge y se obtiene el mínimo óptimo necesario.

Con esto acaba básicamente la explicación de cómo se calcula la predicción en este proceso iterativo de clasificación binaria.

Predicción final

Una vez que una red neuronal ha sido entrenada se puede utilizar para hacer predicciones. Puede realizar predicciones sobre los datos de prueba o validación para estimar la habilidad del modelo en datos no vistos conocidos. También puede implementarlo operativamente y usarlo para realizar predicciones continuamente. La topología de red y el conjunto final de ponderaciones es todo lo que se necesita guardar del modelo. Las predicciones se realizan proporcionando la entrada a la red y realizando un último proceso de propagación hacia adelante que le permite generar una salida que puede usar como predicción.

Ya se conoce cómo calcular de manera teórica todo el proceso de predicción en un problema de clasificación binaria, pero realmente en la práctica todos estos cálculos y pasos se efectúan mediante un programa desarrollado desde un entorno y lenguaje de programación concretos.

Como se expuso anteriormente, se ha usado Python como lenguaje de programación para la realización de este proyecto.

Dentro de Python se presenta Keras, una API diseñada para facilitar el uso de ciertas librerías necesarias para el desarrollo de redes neuronales y aprendizaje automático y profundo.

2.4.2 Keras y Theano

Keras es una API de redes neuronales de alto nivel, escrita en Python. Fue desarrollada con el enfoque de permitir una rápida y eficiente experimentación, es decir, poder pasar de la idea al resultado con el menor retraso posible, ya que, esto es clave para hacer una buena investigación de un problema.

Las principales características de Keras pueden ser:

- Para facilitar el proceso, permite un rápido y sencillo prototipado del modelo.
- Está preparado para llevar a cabo tanto redes convolucionales como recurrentes, así como combinaciones entre ambas.
- Se ejecuta perfectamente en CPU y GPU
- Es compatible con Python 2.7 en adelante.
- Proporciona comentarios claros y procesables sobre el error cometido por el usuario.
- Modularidad**.** Un modelo se entiende como una secuencia de módulos independientes totalmente configurables que se pueden conectar con la menor cantidad de restricciones posible. En particular, las capas neuronales, las funciones de costo, los optimizadores, los esquemas de inicialización, las funciones de activación y los esquemas de regularización son módulos independientes que se pueden combinar para crear nuevos modelos.

- No hay archivos de configuración de modelos separados en un formato declarativo. Los modelos se describen en el código Python, que es compacto, más fácil de depurar y permite la facilidad de extensibilidad.

Keras es una API ligera y en lugar de proporcionar una implementación de las operaciones matemáticas necesarias para el aprendizaje profundo, proporciona una interfaz coherente de librerías numéricas eficientes llamadas backends.

Los backends más usados y conocidos pueden ser tanto Tensorflow como Theano, entre otros. En este caso se ha trabajado con el segundo de ellos por razones de compatibilidad, es decir, **Theano**.

Theano se trata de una librería de Python creada para mejorar la rapidez de la computación matemática pudiendo ejecutarse sobre el CPU o GPU.

Esta es una herramienta clave para el desarrollo de problemas de aprendizaje automático y profundo en Python, ya que, ayuda y complementa a la API de Keras para la creación de los modelos necesarios.

Theano fue un proyecto de código abierto lanzado bajo la licencia BSD y fue desarrollado por el grupo LISA (ahora MILA) de la Universidad de Montreal, Quebec, Canadá. Su nombre está relacionado con un matemático griego.

En el fondo Theano es un compilador para expresiones matemáticas en Python. Sabe cómo tomar sus estructuras y convertirlas en código muy eficiente que utiliza NumPy, librerías nativas eficientes como BLAS y código nativo para ejecutarse lo más rápido posible en CPU o GPU. Utiliza una serie de optimizaciones de código inteligentes para exprimir tanto rendimiento como sea posible de su hardware.

Theano fue diseñado específicamente para manejar los tipos de cálculo requeridos para algoritmos de redes neuronales grandes utilizados en el aprendizaje profundo. Fue una de las primeras bibliotecas de su tipo (desarrollo iniciado en 2007) y se considera un estándar de la industria para la investigación y el desarrollo del aprendizaje profundo.

2.5 ENTRENAMIENTO

PREPARACIÓN DE LOS DATOS

Primeramente, se deben preparar sus datos para el entrenamiento en una red neuronal. Los datos deben ser numéricos, por ejemplo, valores reales. Si se tienen datos categóricos, como por ejemplo un atributo de sexo con los valores masculino y femenino, pueden convertirse en una representación de valor real denominada codificación en caliente (en inglés, “one hot encoding”). Aquí es donde se agrega una nueva columna para cada valor de clase (dos columnas en el caso de sexo de hombre y mujer) y un 0 o 1 para cada fila dependiendo del valor de clase para esa fila.

Esta misma codificación en caliente se puede utilizar en la variable de salida en problemas de clasificación con más de una clase. Esto crearía un vector binario a partir de una sola columna que sería fácil de comparar directamente con la salida de la neurona en la capa de salida de la red, que como se describió anteriormente, generaría un valor para cada clase.

Las redes neuronales requieren que la entrada se escale de forma coherente. Puede ser escalado al rango entre 0 y 1 llamado normalización. Otra técnica popular es estandarizarla para que la distribución de cada columna tenga la media de cero y la desviación estándar de 1.

BASE DE DATOS DE ENTRENAMIENTO

Para la realización del proceso de entrenamiento es necesario el uso de una cantidad considerable de datos de entrada a la red, ya que, cuanto más número total de datos sean introducidos en dicho entrenamiento, la red será capaz de aprender de una mejor manera y contemplar todas, o las máximas configuraciones posibles de aprendizaje.

Por eso, debido a que el proyecto en cuestión trata sobre un algoritmo de transcripción musical para piano concretamente, por tanto, la base de datos necesaria estará compuesta por numerosos archivos de notas distintas e interpretadas por diversos tipos de pianos.

Esta enorme cantidad de información proviene de una base de datos descargada de internet llamada MAPS, cuyo nombre viene de "MIDI Aligned Piano Sounds". Se trata de una base almacenada con infinidad de fragmentos de notas musicales distribuidas de varias formas, con distintas duraciones, en distintas condiciones y cambiando la dinámica con respecto unas de otras para que exista mayor variedad de datos.

PRINCIPALES CARACTERÍSTICAS DE MAPS.

MAPS proporciona grabaciones con calidad similar a la de un CD, es decir, una frecuencia de muestreo de audio en estéreo de 44100 kHz y usa 16 bits.

La parte posiblemente más útil de esta base de datos es que cada uno de los archivos MIDI alineados está asociado a un archivo de texto en el cual se incluyen 3 parámetros fundamentales para facilitar su entendimiento y uso, que son: **tiempo de inicio de un pitch** (onset times), **tiempo de final de un pitch** (offset times) y **el pitch** que corresponda en cada intervalo de tiempo.

El tamaño total de la base de datos es aproximadamente de unos 40 Gigabytes, es decir, más o menos unas 65 horas de audio grabado, aunque no se ha usado todos los archivos de esta, pero si una gran parte de ella.

La base de datos está disponible de manera libre y gratuita, pero bajo una licencia de "Creative Commons".

Toda esta cantidad enorme de datos y sus respectivos archivos de información de texto son proporcionados gracias a algunos procesos de generación automática, que son formados a partir de la síntesis de audio en archivos MIDI.

El uso de un Disklavier (piano MIDI) y de un software de síntesis de alta calidad basado en bibliotecas de muestras permitió un intercambio satisfactorio entre la calidad de los sonidos y el consumo de tiempo necesario para producir tal cantidad de sonidos anotados.

El contenido de MAPS está dividido en 4 secciones:

- **ISOL**: se trata de un conjunto de archivos que envuelven todo lo que es el audio monofónico de la base de datos.

Puede ser muy útil si se tiene como objetivo testear algoritmos estimativos de tono único o entrenar algoritmos de múltiples tonos cuando se requieren tonos aislados.

Todos los archivos comparten la misma composición del nombre formada por varios campos: ***MAPS_ISOL_ps_i0_Ss_Mm_instrName.wav***

- ***ps***, estilo de reproducción, puede ser:
  - NO: notas tocadas de manera normal de 2 segundos de duración.
  - LG: notas largas (la duración en este caso puede variar desde 3 segundos en notas con el pitch más agudo hasta 20 segundos en notas con el pitch más grave).
  - ST: staccato.
  - RE: la misma nota se va repitiendo cada vez más rápido, desde aproximadamente 1.4 hasta 13.5 notas por segundo.
  - CHd: escalas cromáticas ascendentes y descendentes, con varias duraciones en las notas, indicadas por d.
  - Tri: trinos, cada vez más rápidos, de medio tono (i = 1) o de un tono entero (i = 2), desde 2.8 hasta 32 notas por segundo.
- ***i0***, el volumen (dinámica), puede variar:
  - P: piano
  - M: mezzo-forte
  - F: forte
- ***s***, variable binaria que determina cuando el pedal esta presionado (s = 1) y cuando no (s = 0). Por tanto, se usa el pedal en el 50% de los casos, presionando este siempre 300ms antes del comienzo de la secuencia y soltándolo 300ms tras el final.
- ***instrName,*** se trata del campo que determina en que condiciones y con que instrumento se ha interpreta la reproducción en cuestión. Cada una de las opciones están codificadas en la siguiente tabla mediante la etiqueta "Code":

Code	Instrument model	Recording conditions	Real instrument or software
StbgTGd2	Hybrid	Software default	The Grand 2 (Steinberg)
AkPnBsdf	Boesendorfer 290 Imperial	church	Akoustik Piano (Native Instruments)
AkPnBcht	Bechstein D 280	concert hall	Akoustik Piano (Native Instruments)
AkPnCGdD	Concert Grand D	studio	Akoustik Piano (Native Instruments)
AkPnStgb	Steingraeber 130 (upright)	jazz club	Akoustik Piano (Native Instruments)
SptkBGAm	Steinway D	"Ambient"	The Black Grand (Sampletekk)
SptkBGCl	Steinway D	"Close"	The Black Grand (Sampletekk)
ENSTDkAm	Yamaha Disklavier Mark III (upright)	"Ambient"	Real piano (Disklavier)
ENSTDkCl	Yamaha Disklavier Mark III (upright)	"Close"	Real piano (Disklavier)

Tabla i. Características de los archivos de audio según su etiqueta

- ***m,*** pitch del archivo de audio codificado entre [21 - 108] que se corresponden con las 88 notas con las que se trabaja en formato MIDI, siendo la nota 21 la más grave y, por tanto, la nota 108 la más aguda.

- **RAND**: conjunto formado por acordes de notas aleatorias, desde acordes formados por 2 notas hasta 7 simultáneamente.

  Fue diseñado principalmente para evaluar los algoritmos de estimación multi-pitch de manera objetiva, sin ningún conocimiento musical a priori.

  Cada acorde almacenado se nombre siguiendo las mismas notaciones, de la siguiente forma: ***MAPS_RAND_Px_Mm1-m2_Ii1-i2_Ss_nn_instrName.wav***

  Donde:

  - ***x,*** varía de 2 a 7 siendo el nivel de polifonía.
  - El rango de pitch ***m1-m2*** puede ser [21-108] o [36-95]. El primer rango consiste en el rango completo de un piano (7 + ¼ octavas), mientras que el segundo se extiende sobre las 5 octavas centradas.
  - El volumen o intensidad se escoge independientemente para cada nota, en dos rangos posibles:
    - 60-68 → mezzo-forte, que puede representar un acorde típico en que sus tonos tienen intensidades bastante similares.
    - 32-96 → piano – forte, que puede reflejar el contenido polifónico cuando diversas líneas melódicas son interpretadas, resultando que se crean niveles de intensidad heterogéneos.
  - ***s,*** como ya se ha dicho denota el uso o no del pedal.
  - ***n,*** denota el índice de resultado.

- **UCHO**: este conjunto trata sobre acordes típicamente usados en composiciones de música del oeste.
  En la base de datos creada para este proyecto no se han tomado fragmentos y archivos de este conjunto.

- **MUS**: en este caso, este conjunto consiste en piezas míticas de música clásica para piano.

  Esta sección proporciona piezas de música generadas a partir de archivos MIDI estándar disponibles en Internet bajo una licencia Creative Commons. Estos archivos de alta calidad han sido cuidadosamente escritos a mano para obtener un tipo de interpretación musical como un archivo MIDI. La ubicación de la nota, la duración y el volumen han sido ajustados a mano por el creador de la base de datos MIDI. Alrededor de 238 piezas de música clásica y tradicional fueron creadas y están realmente disponibles en la base de datos de MAPS (http://www.piano-midi.de).

Por tanto, teniendo a disposición esta base de datos se han reordenado y reformado muchos de los archivos citados anteriormente para llegar a conseguir una base de datos final dividida en 2 sub base de datos.

Una de ellas estará constituida únicamente por archivos de sonidos monofónicos, es decir, notas aisladas escogidas dentro de la sección ISOL de MAPS, mientras que la otra parte de

la base de datos estará formada por archivos con acordes aleatorios de la sección RAND de MAPS.

BASE DE DATOS MONOFÓNICA

Como se ha dicho, a partir de extractos del conjunto de datos llamado ISOL en MAPS se va a configurar una base de datos con sonidos con un solo "pitch" que nos será de utilidad para el entrenamiento de la red neuronal del proyecto.

Para su realización, primeramente, se reformarán los archivos necesarios de manera que, mediante técnicas de procesamiento de audio y gracias a la ayuda de los archivos de texto que contienen la duración de cada tono, cada sonido monofónico quedará almacenado de forma independiente en pequeños archivos, cada uno de ellos asociado a una nota MIDI en concreto que como se ha dicho antes estarán comprendidos los tonos entre el mínimo 21 y el máximo 108.

De esta forma, se tendrán cientos de mini archivos formados por las distintas 88 notas totales e interpretadas por diversos tipos de pianos y derivados y en varias condiciones.

Por lo que, se realizará una ordenación de los mismos, en la que todos los archivos que estén compuestos por el mismo tono se reagruparán obteniendo de esta forma 88 agrupaciones de sonidos diferentes.

Una vez que se tienen divididos y ordenados estos 88 grupos con sus respectivos archivos, se concatenan todos los archivos de cada grupo, de manera aleatoria, consiguiendo así una base de datos final con 88 únicos y pesados archivos que constituyen a cada una de las notas de un piano general de $7^{¼}$ octavas.

Todo este procesamiento y reformación de audio se realiza mediante algoritmos codificados y ejecutados en MATLAB ya que es una herramienta muy potente para este tipo de procesamiento.

BASE DE DATOS POLIFÓNICA

En este caso, se ha trabajado con archivos pertenecientes al conjunto de datos llamado RAND de MAPS. Estos archivos como ya se ha explicado, se tratan de sonidos polifónicos formados con acordes con configuraciones interválicas aleatorias, es decir, no sigue ningún tipo de diferencia entre los intervalos típica de ninguna tonalidad en concreto.

Estos acordes pueden estar constituidos desde 2 tonos simultáneamente hasta agrupaciones de 7 tonos al mismo tiempo.

Por lo tanto, a partir de estos archivos y sus correspondientes archivos de texto asociados, mediante MATLAB como con la base de datos monofónica, se ha procedido a arreglar estos archivos de manera que sean más útiles y óptimos para la realización del entrenamiento.

Lo que se ha llevado a cabo en este caso, ha sido reformar cada uno de los acordes de cada archivo, con lo que conseguir que el sonido del acorde comience justo en el comienzo y cese justo en el final de las muestras del archivo, ya que de esta manera se evita introducir mucho silencio e información que no es de tanta utilidad. Para ello, se trabaja y se procesa apoyándose en el archivo de texto con la información de cada acorde.

Además, en la información de cada sonido también aparece, como se detalló previamente, cuáles son los tonos (pitches) que se encuentran en cada archivo, por lo que, gracias a esto se ha creado y exportado de manera paralela una salida de cada uno de los sonidos. Esta salida es realmente una matriz que consta de 88 filas correspondientes a cada una de las notas y tantas columnas como muestras temporales contenga el archivo.

De esta forma, cada vez que un sonido está formado por unos pitches determinados, se activan (se pone a 1) únicamente en ese instante temporal (número de muestra) las filas que están asociadas a esos tonos. Los demás componentes de esa columna que corresponde con un instante de tiempo quedan desactivados, es decir, a 0.

Por tanto, se obtiene finalmente una base de datos polifónica en la que existen seis grupos de datos, siendo estos acordes desde 2 notas hasta acordes de 7 notas, teniendo también asociadas las correspondientes salidas a cada uno de los acordes.

Una vez que se tienen preparadas las dos sub bases de datos se agrupa toda la cantidad de información obtenida para pasar a derivarla en 3 secciones de datos:

- **Datos de entrenamiento:** es la parte de la información que está destinada al entrenamiento y aprendizaje del algoritmo de transcripción musical.

  Cuanto mayor sea la cantidad de datos que sea introducida en la red, mejores resultados podrán ser obtenidos debido a que tiene más capacidad de configuraciones que tomar y tener en cuenta a la hora del entrenamiento.

- **Datos de validación**: una parte muy importante en el entrenamiento de la red es la validación de los resultados, ya que, con este proceso se calculan los errores cometidos en el aprendizaje cada vez que se realiza la propagación hacia adelante y se obtiene una predicción. Por tanto, se actúa en consecuencia para conseguir que la velocidad de aprendizaje sea máxima y no se den rodeos actualizando pesos que no son óptimos o necesarios.
- **Datos de testeo**: se trata de una pequeña cantidad de datos, en comparación con los de entrenamiento, que es usada y propagada únicamente hacia adelante a lo largo de la red para obtener una predicción de esos datos que será la salida final de predicción del sistema.

  En el caso de entrenamiento, este tipo de datos solo se usa para comprobar que se tiene un resultado aceptable de la predicción, pero en realidad, a la hora de ejecutar el programa una vez que ya ha sido entrenado previamente, este conjunto de datos será la cantidad de información que se requiera predecir.

Por ejemplo, en este proyecto, los datos de testeo se corresponden con pistas de audio escogidas para que sean predichas para posteriormente ser usadas en la aplicación final del sistema.

Conocido esto, esta derivación de los datos no es equitativa, es decir, no se dividen en partes porcentuales iguales, ya que, como se ha expuesto se necesitan una cantidad de datos bastante mayor en el proceso de entrenamiento que en el resto.

Por tanto, se ha distribuido de manera que, un 60% de la base de datos total conforman los datos del entrenamiento del sistema neuronal.

Por otro lado, el 40% restante se ha dividido en dos partes iguales para los datos de validación y testeo, es decir, un 20% para cada uno de ellos.

Antes de continuar, explicando y detallando las evaluaciones realizadas en el sistema de entrenamiento es necesario aclarar que todos estos datos de entrada a la red, ya sean los de entrenamiento como los de validación o testeo, antes de ser introducidos en la red deben de ser procesados y transformados, mediante la transformada constant-Q citada y detallada con anterioridad, ya que hay que el procesamiento neuronal se realiza en el dominio espectral y no en el dominio temporal.

De esta manera, los datos de entrada pasan a tener unas dimensiones de 252 coeficientes por el número de tramas en las que se compongan finalmente cada archivo.

Es necesario notar que al realizar la transformada, cada trama resultante, compuesta por 252 coeficientes relacionados con su forma espectral, ha sido formada por 512 muestras temporales, las correspondientes a cada tramo de tiempo.

Una vez que se obtienen todos los datos a usar transformados en el dominio de la frecuencia, el último paso que siguen estos antes de su entrada al entrenamiento es la normalización de los mismos, ya que, la matriz de datos resultante de la transformada con los coeficientes espectrales en cada instante no se encuentra precisamente entre valores de 0 y 1, por lo que, esta normalización ayuda al entendimiento y coste del aprendizaje del sistema.

Para ello, se ha usado un algoritmo en el lenguaje Python que normaliza los datos obtenidos con la normalización típica de sustracción de la media y división de la desviación típica, dejando los datos con una media cercana a 0 y una desviación típica aproximadamente 1 , pero además a su favor, no lo realiza de golpe, es decir, realizando este proceso a la cantidad de datos total, si no que se va realizando por tramas y posteriormente se almacenan para poder ser actualizados estos parámetros estadísticos.

Este tipo de normalización a tramos ayuda a que no existan grandes variaciones en el valor de los datos, si no que se ajustan y se adaptan los datos normalizados en valores que siguen secuencias parecidas.

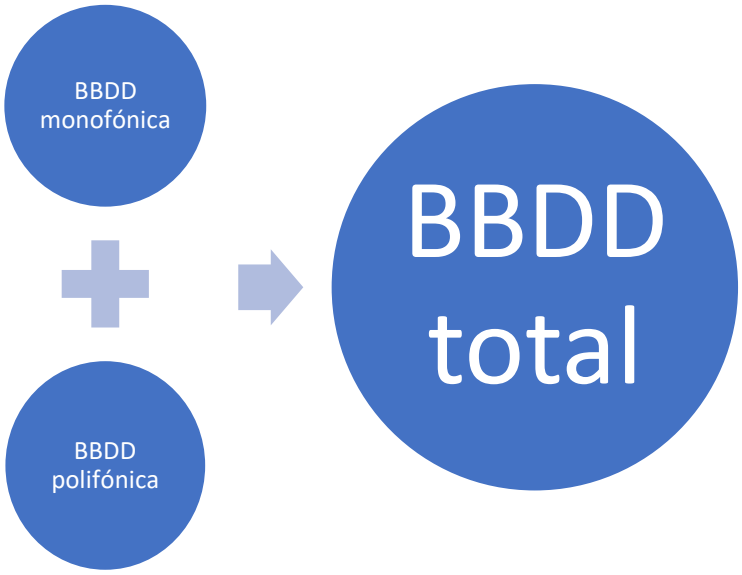
EVALUACIÓN

Una vez que se conoce de manera bastante completa de que tratan las redes neuronales, como pueden ser configuradas y teniendo también conocimiento de su base de datos necesaria, se ha propuesto una evaluación de distintas configuraciones de redes, ya sea variando las capas internas como la cantidad o el tipo de datos introducidos, así también como, el tipo de tecnología de red neuronal, centrándose en una comparación entre redes neuronales profundas (DNN's) y redes neuronales convolucionales (CNN's).

En primer lugar, hay que aclarar que tras su previa investigación se va a trabajar con una base de datos formada por la suma de todos los datos monofónicos junto con los polifónicos, ya que, la red necesita ser provista de todos ellos para que sea capaz de aprender también las configuraciones melódicas y escalas cromáticas y no solo acordes.

Por lo que, antes de comenzar el entrenamiento, el programa carga ambos conjuntos de datos, los concatena y se normalizan teniendo en cuenta, obviamente, todo el cúmulo de datos existente.

Ilustración xxi. Esquema sobre la síntesis de la base de datos total



Tras esto, se ha realizado un estudio dentro de un tipo de red basado en perceptrón multicapa, variando la configuración del modelo, modificando el número de capas ocultadas usadas, así como, el número de unidades neuronales que tendrán cada una de ellas.

Para el diseño de la red se han usado funciones de activación ReLU para todas las capas internas y función sigmoide para la última capa, es decir, la de predicción. También, se ha escogido un optimizador de tipo ADAM y una función de pérdidas basada en la entropía cruzada binaria, ya que es la más común en este tipo de problemas.

En el proceso de evaluación, como es de suponer, se evalúa cada tipo de configuración a razón de un parámetro calculado al final de cada proceso hacia delante de la red. El cálculo de este valor de precisión se realiza a través de un modelo programado en Python que da como salida dos parámetros:

- ✓ f1_framewise → se trata de un valor basado en un error relativo, comprendido entre [0 - 1] que expresa cuanto se parece la matriz original de salida con respecto a la matriz predicha en su validación, siendo 1 el máximo valor de precisión posible.
- ✓ error_tot → mide el error cometido con un valor comprendido entre [0-1] siendo 0 el menor error posible cometido.

Para permitir un aprendizaje más óptimo, en el código creado para el modelo de entrenamiento y validación se ha establecido un contador, que se llama "patience", paciencia en inglés. Este original contador lo que hace es que cada 3 incrementos, calcula estos parámetros que miden la precisión (f1_framewise, error_tot) y los compara con los calculados la vez anterior.

Si los calculados recientemente tienen un mejor resultado que los que fuesen mejor previamente se actualizan pasando estos a ser los resultados válidos y el modelo guarda y actualiza también todos los parámetros y ponderaciones de ese proceso de entrenamiento para actualizarlos en la propagación hacia atrás siguiente. También en este caso se inicializará el contador a 0.

Por otro lado, si los resultados obtenidos una vez que han sido calculados son peores que otros anteriormente obtenidos, entonces no se cambiará nada y no se actualizará el modelo tampoco, quedándose con los valores obtenidos en un ejemplo de entrenamiento y validación anterior.

El contador de paciencia se irá incrementando de 3 en 3 cada vez que calcula los valores de precisión mientras no se mejore el resultado de la predicción hasta que se alcance un valor límite de paciencia, por ejemplo, establecido en 20.

El programa también va guardando y actualizando el modelo en un archivo con formato ".h5", que se trata de un formato especial para almacenamiento de pesos de un modelo de redes neuronales.

A su vez, esta función de seguimiento de validación y predicción también va creando, de manera paralela al cálculo de los parámetros de precisión, una figura en formato PNG que representa como van sucediendo estos parámetros a lo largo del periodo total de entrenamiento y da información del comportamiento de cada configuración del modelo creando una curva que expresa como va disminuyendo el coste de la red y otro apartado con las curvas de f1_framewise y error_tot.

Observando las gráficas que aparecen a continuación, se puede deducir de manera intuitiva cual o cuales de los modelos trabajan y aprenden de mejor forma.

Se va a exponer varias configuraciones probadas para dar finalmente con el modelo más óptimo y así, adoptarlo en el sistema final para el desarrollo del proyecto al completo.

Características principales de las redes "DNN" en evaluación

Antes de mostrar los resultados gráficamente de la evaluación es necesario aclarar todos los parámetros establecidos para configurar la red en cada caso.

La evaluación se ha realizado alterando el número de capas ocultas de la red neuronal, desde 2 capas en el menor de los casos, hasta 4.

El número total de unidades neuronales en cada capa también va a ser modificado en cada prueba, probando prácticamente con todas las configuraciones posibles, tomando 3 valores distintos: 250, 500, 1000 uds.

En todas las redes evaluadas se ha usado activación ReLU en la primeras capas ocultas y activación sigmoide únicamente en la capa más externa.

El vector de entrada a las redes va a tener una longitud de 252, correspondiente a cada uno de los coeficientes exportados por cada uno de los filtros de la transformación espectral.

La matriz de salida predicha estará formada, en todos los casos, por 88 valores en un eje, asociados a cada una de las notas predichas por el sistema de clasificación y el número de tramas que compone el espectro estimado en el otro eje.

Por otro lado, la función de coste con la que se calcula la predicción está basada en la función de entropía cruzada binaria, detallada en el apartado 2.4.

También, es necesario resaltar el uso de un optimizador de tipo ADAM y, además, para evitar sobre entrenamiento, a modo de regulación se ha implementado en cada capa un valor de "Dropout" igual a 0.3.

Se ha aplicado, de igual modo en cada capa interna, un módulo especifico de la librería de "Keras" llamado Batch Normalization (Normalización por lote), que ayuda a la correcta estandarización y ajuste de los datos al problema.

Características principales de las redes "CNN" en evaluación

En cuanto a la red convolucional, se conforma de dos secciones independientes.

La primera se trata de la capa convolucional bidimensional, configurada con 50 filtros y un tamaño de kernel de (1×25). Tras esta capa superior se aplica la máscara (maxpooling) con una dimensión de (1×3) que se recorre por la imagen espectral.

Con respecto a la segunda sección, se trata de una configuración similar a la de DNN por lo que se ha establecido la mejor configuración obtenida en este caso.

➢ CONFIGURACIONES DE RED DNN CON 2 CAPAS INTERNAS:

Ilustración xxii. Configuración de red - 1000 + 500 unidades

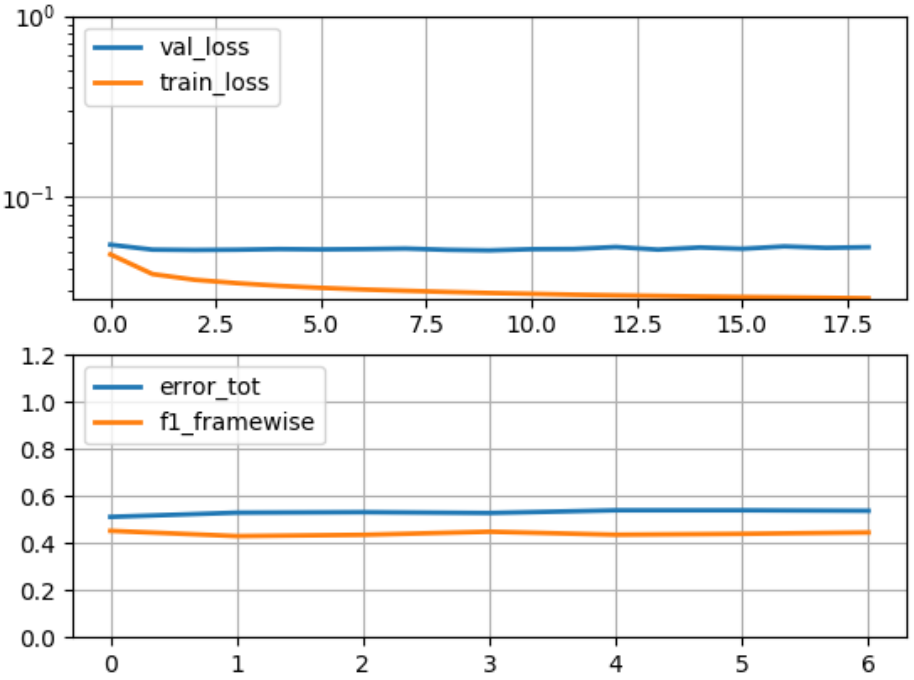


Ilustración xxiii. Configuración de red - 1000 + 250 unidades

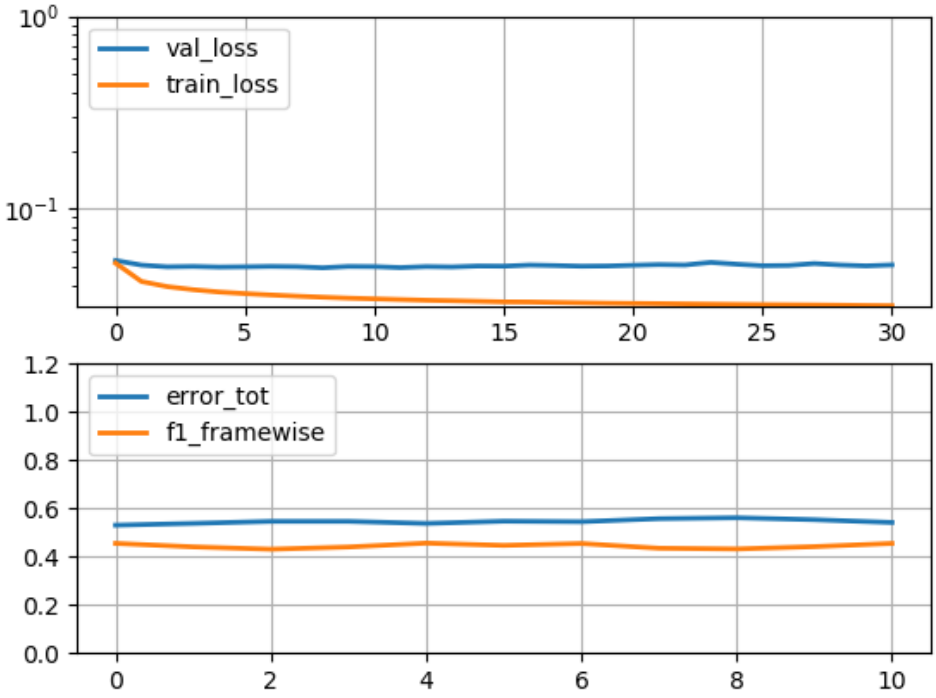


Ilustración xxiv. Configuración de red - 500 + 500 unidades

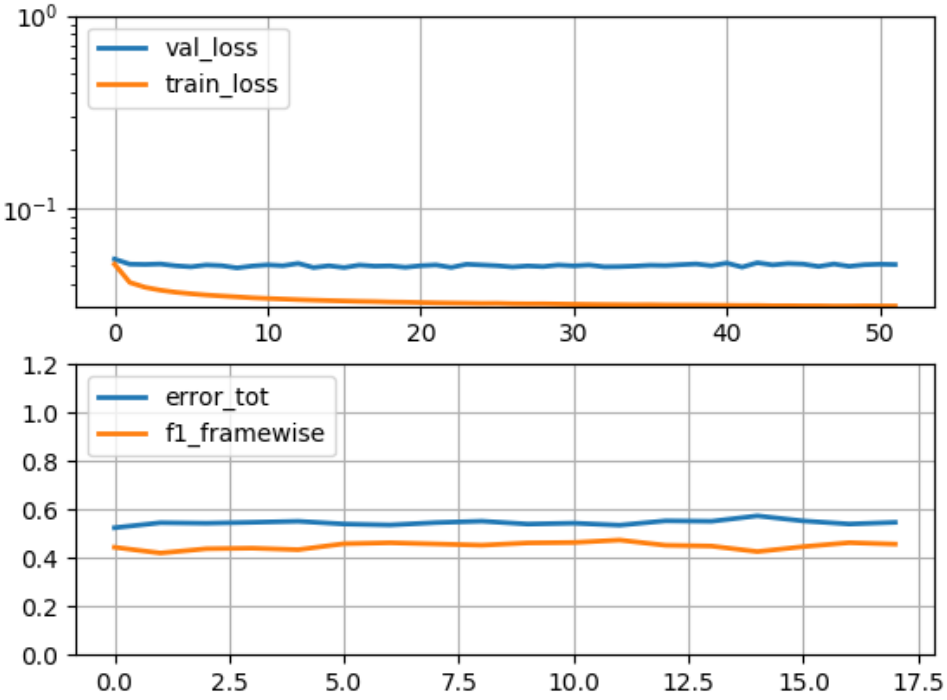


Ilustración xxv. Configuración de red - 500 + 250 unidades

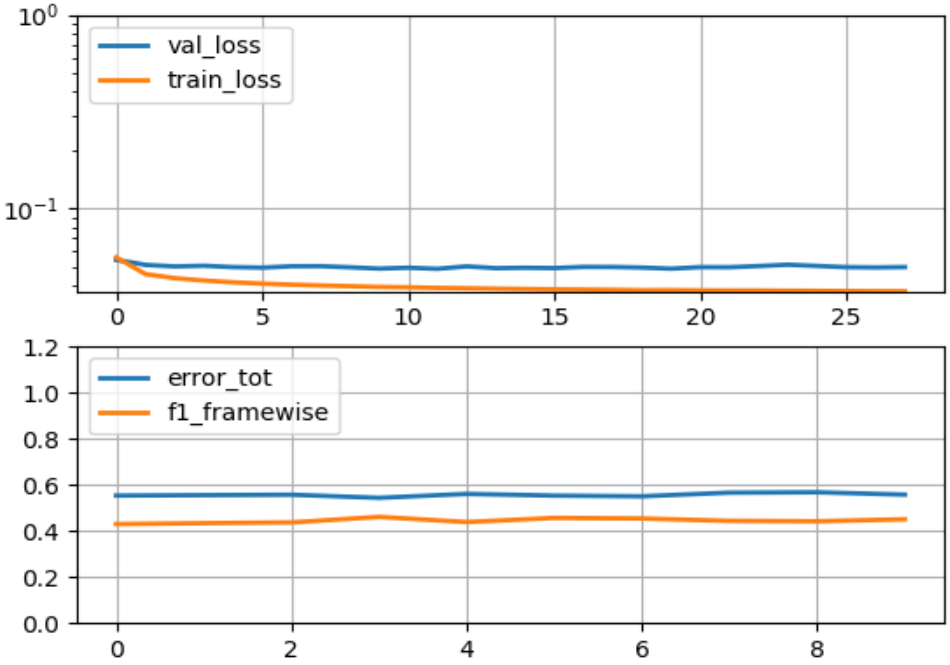


Ilustración xxvi. Configuración de red - 250 + 500 unidades

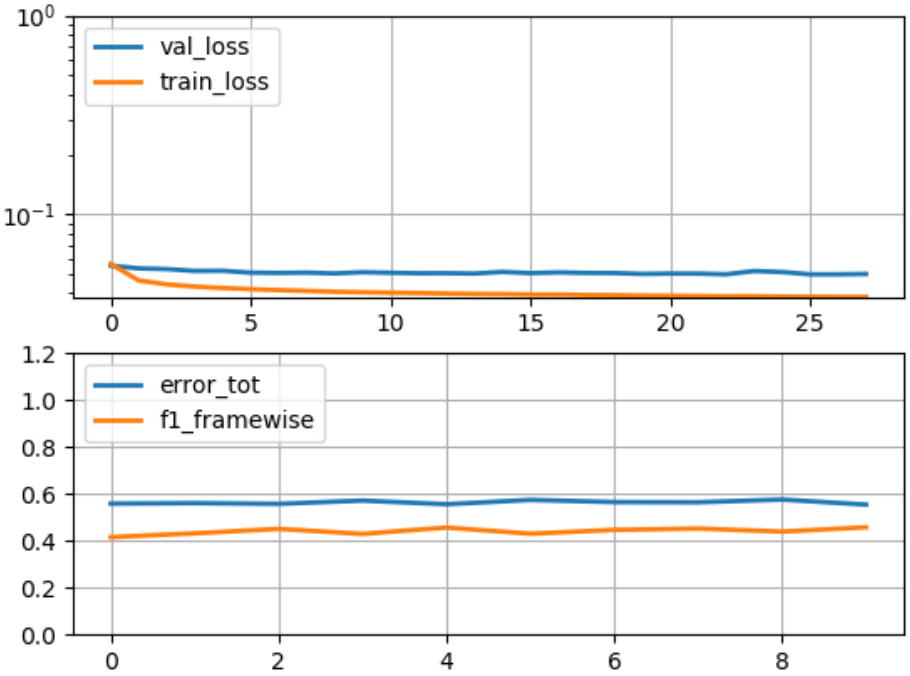
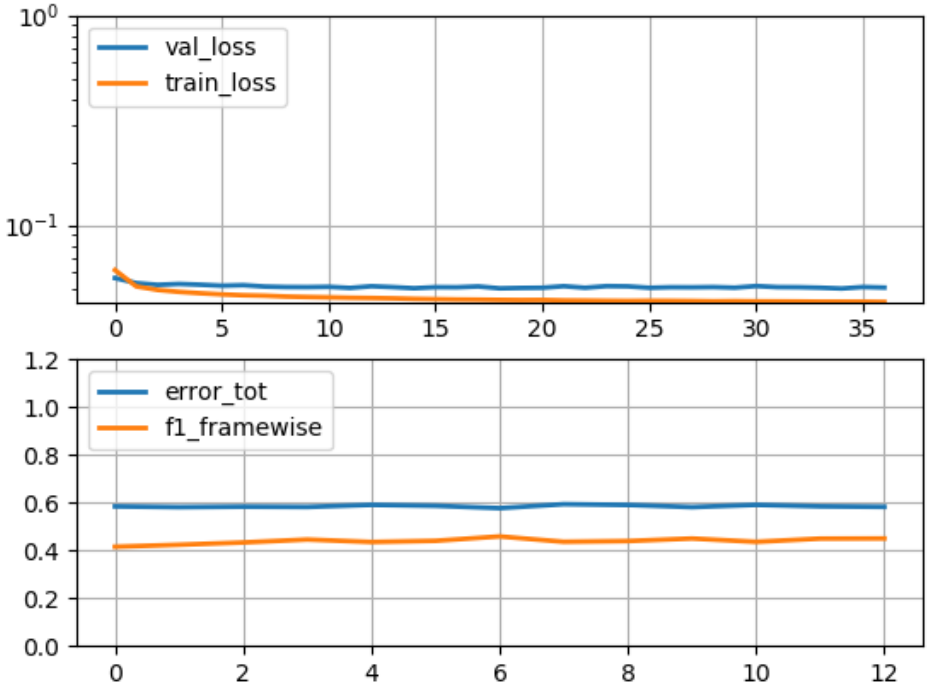


Ilustración xxvii. Configuración de red - 250 + 250 unidades



➢ CONFIGURACIONES DE RED DNN CON 3 CAPAS INTERNAS

Ilustración xxviii. Configuración de red - 1000 + 500 + 250 unidades

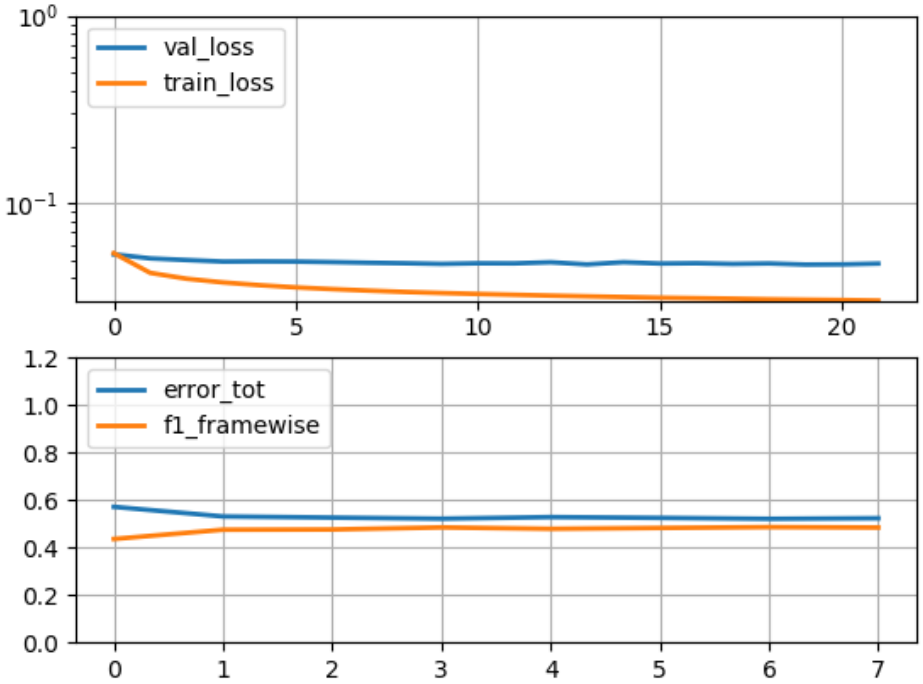


Ilustración xxix. Configuración de red - 1000 + 250 + 250 unidades

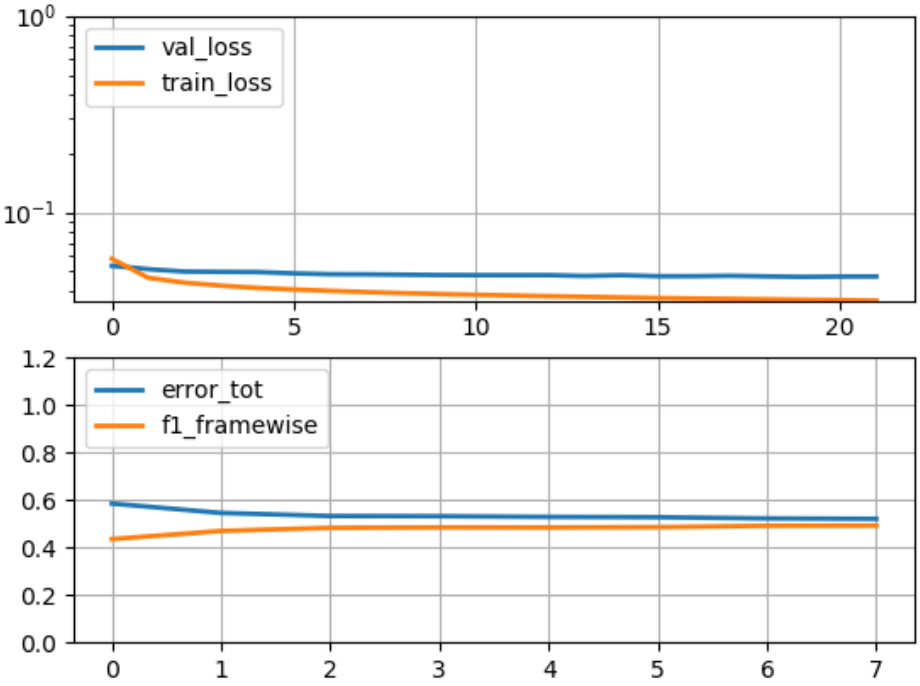


Ilustración xxx. Configuración de red - 500 + 500 + 250 unidades

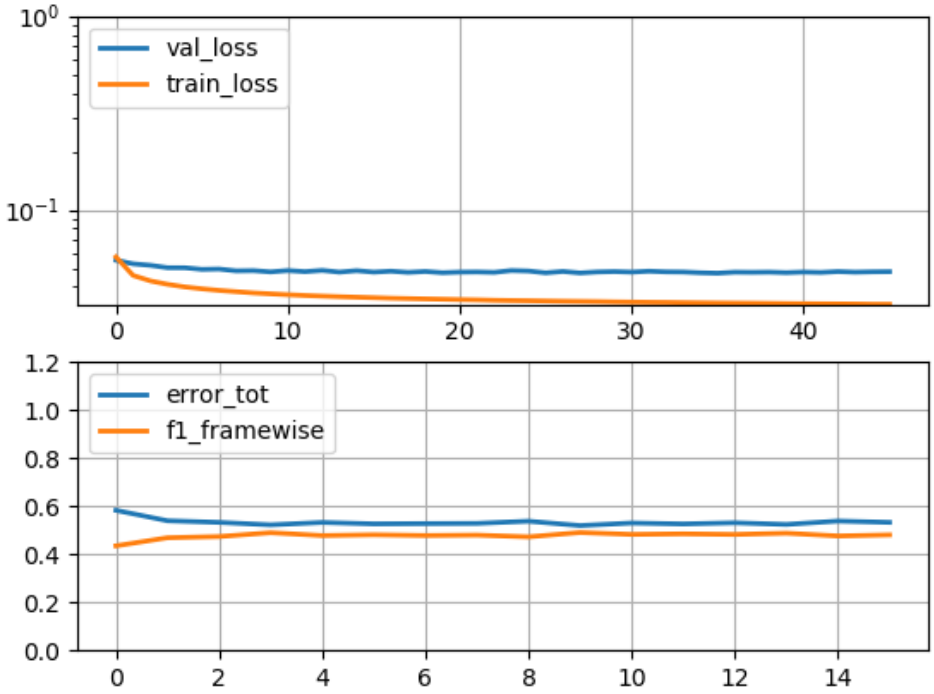


Ilustración xxxi. Configuración de red - 500 + 250 + 250 unidades

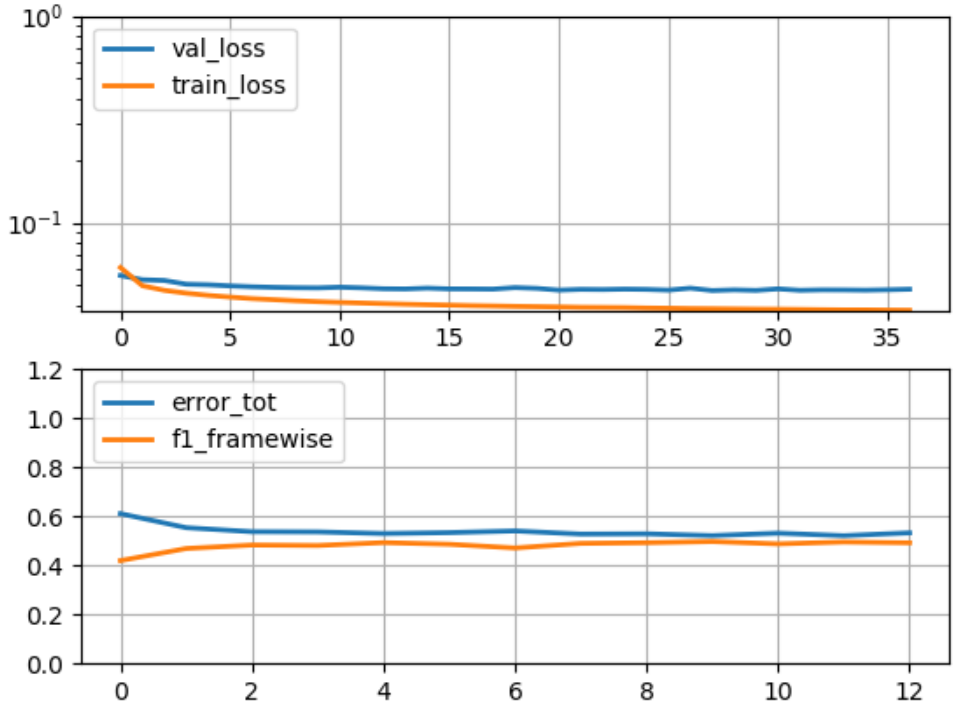


Ilustración xxxii. Configuración de red - 250 + 500 + 250 unidades

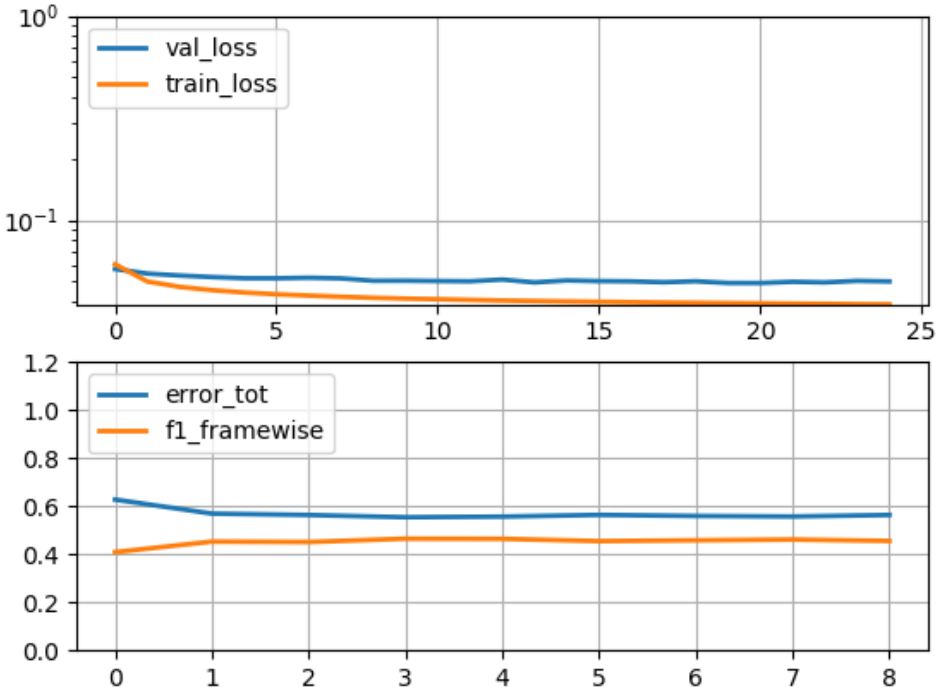
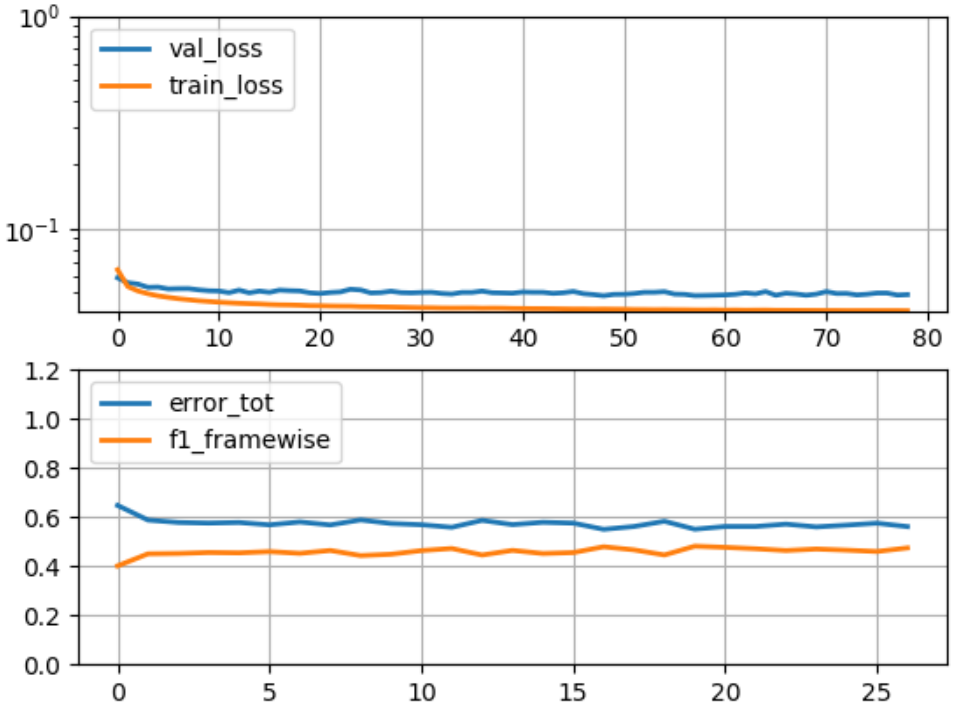
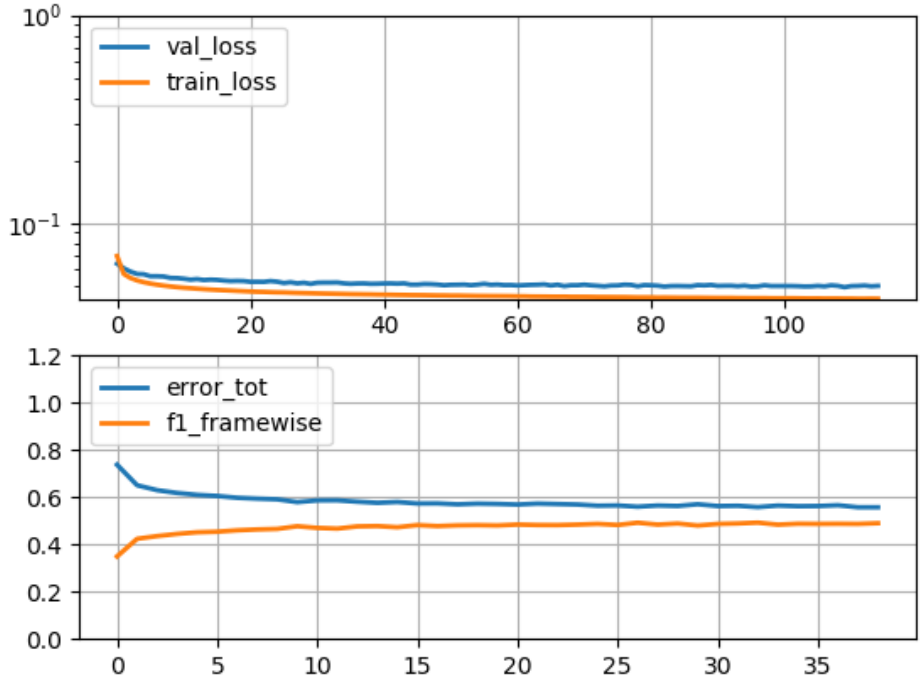


Ilustración xxxiii. Configuración de red - 250 + 250 + 250 unidades



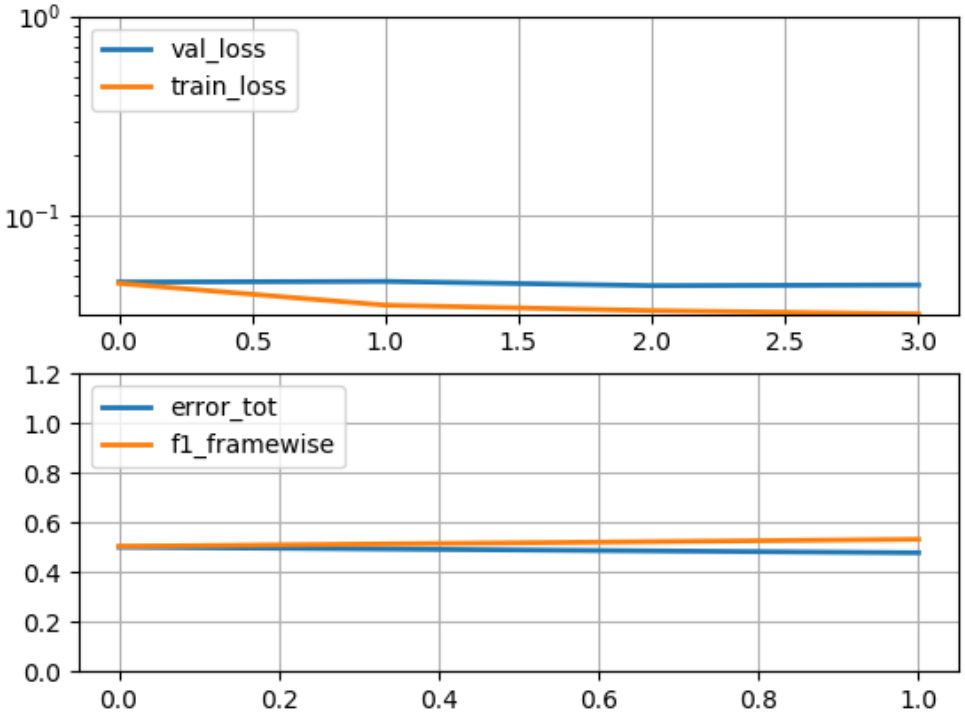
- CONFIGURACION DE RED DNN CON 4 CAPAS INTERNAS

Ilustración xxxiv. Configuración de red - 500 + 250 + 250 + 125



- CONFIGURACIÓN DE RED CNN

Ilustración xxxv. Configuración de red convolucional



Por tanto, haciendo una breve recapacitación y consideración de las gráficas de los diversos modelos se puede deducir que:

El modelo que mejor respuesta de aprendizaje presenta será el configurado en la figura ix, ya que, se trata de un modelo formado por 3 capas internas, además de las capas periféricas de entrada y salida, con un número de unidades por capa (propagándose de entrada a salida) de 500 – 250 – 250 unidades neuronales, respectivamente.

Este modelo presenta, como se puede ver en su gráfica, un parámetro de precisión aproximadamente de 0.5, siendo el mayor de los obtenidos por todos los modelos. Si se observa en las curvas del coste de entrenamiento y validación también se puede ver que presenta una buena respuesta, tendiendo este coste prácticamente a 0 cuando la red ha sido entrenada.

Se podrían usar otro tipo de configuraciones sin ningún problema presentando buenos resultados casi de igual manera. Por ejemplo, el modelo de 3 capas con 500 – 500 – 250 unidades, da muy buen resultado también, prácticamente similar en cuanto a precisión se refiere, pero es un modelo un poco más pesado por lo que esa es la razón por la que se ha elegido el modelo anterior. Del mismo modo, el modelo de 4 capas internas también proporciona unos resultados de predicción similares a estos anteriores, pero este modelo será más pesado aún, por lo que, para ser más eficientes en la forma de trabajo tampoco se usará para el sistema final.

Por otro lado, como se puede apreciar claramente, el modelo basado en CNN, el cual ha sido configurado en sus capas de DNN con la mejor configuración obtenida en la evaluación anterior, presenta una respuesta de aprendizaje y predicción superior a la red simple DNN. Aun así, no resulta un modelo muy eficiente de uso, ya que, presenta un coste computacional de más 10 veces superior al coste de DNN, por lo que, con los resultados obtenidos con la red DNN será suficiente para un buen funcionamiento.

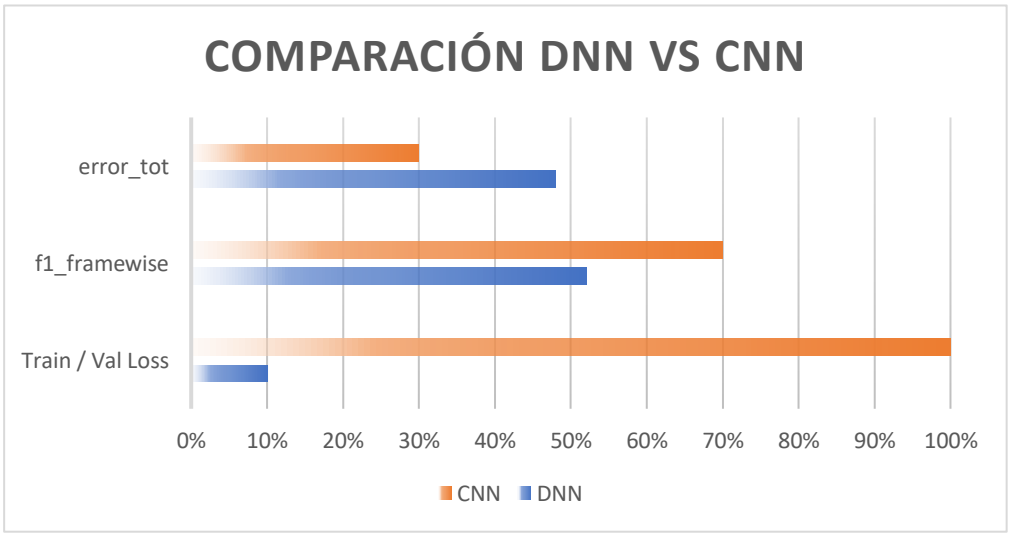


Tabla ii. Comparación DNN vs CNN

3. ALINEAMIENTO

3.1 INTRODUCCIÓN

Las versiones MIDI disponibles de pistas de música populares pueden proporcionar una gran cantidad de datos básicos útiles para tareas de recuperación de información de música incluyendo, por ejemplo, el seguimiento de ritmo / compás, detección de inicio, transcripción automática o separación de fuentes basado en puntuaciones.

Sin embargo, para maximizar su valor, el archivo MIDI dado debe estar alineado en tiempo a su correspondiente archivo de audio. Este problema de alineamiento está estrechamente relacionado con el emparejamiento MIDI-audio, que lo que principalmente busca es identificar que archivo de audio corresponde a un MIDI determinado, perteneciendo ambos archivos a colecciones de datos independientes.

En estos escenarios, es de gran valor producir un alineamiento de confianza, el cual expresa cómo de preciso el contenido de un archivo MIDI coincide con un archivo de audio dado, lo que puede reflejar la calidad de la transcripción.

Existe una amplia variedad de técnicas que hacen que se produzca un alineamiento o un emparejamiento entre MIDI y audio. Para ello, se va a detallar y a usar en este proyecto la técnica más común en este tipo de escenarios, conocida en inglés como, “Dynamic Time Warping” (deformación dinámica del tiempo). Esta técnica, a priori, se propuso para la comparación de declaraciones de voz.

DTW usa programación dinámica para llegar hasta un alineamiento monotónico, tal que la suma de la distancia de coste entre los vectores alineados de características se minimiza. Esta propiedad lo hace ideal para la alineación de audio a MIDI cuando se conoce que el MIDI se trata de una transcripción continua precisa, es decir, sin secciones incorrectas.

Dado que la distancia total entre pares alineados de vectores de características da un valor global único, un buen resultado de DTW será una medida natural que muestra el índice de similitud entre dos secuencias.

A pesar de su uso generalizado, el éxito de DTW depende de su parametrización, así como las opciones de diseño del sistema, como la función de representación y la métrica de distancia local utilizada.

3.2 DTW PARA ALINEAMIENTO

Suponiendo dos secuencias dadas correspondientes de vectores de unas ciertas características, siendo:

$$X \in \mathbb{R}^{M \times D} \quad (89)$$

$$Y \in \mathbb{R}^{N \times D} \quad (90)$$

Donde, *D* es la dimensionalidad característica, mientras que *M y N* son el número de vectores de características de *X e Y,* respectivamente. La técnica DTW produce, por tanto, dos secuencias siempre de forma creciente:

$$p, q \in \mathbb{N}^{L} \quad (91)$$

Las cuales definen el alineamiento óptimo entre *X e Y,* tal que, siendo:

$$p[\boldsymbol{i}] = n \quad (92)$$

$$q[\boldsymbol{i}] = m \quad (93)$$

Esto implica que el m-ésimo vector característico de X debería estar alineado al n-ésimo de Y.

Por lo tanto, resumiendo, en la técnica de deformación dinámica del tiempo, para encontrar las secuencias alineadas *p y q* conlleva a resolver el siguiente problema de minimización:

$$p, q \; = \; arg \min_{p,q} \sum_{i=1}^{L} \left\| X\left[p[i]\right] \; - \; Y\left[q[i]\right] \right\|_2^2 + \Phi(\boldsymbol{i}) \quad (94)$$

Donde $\Phi(i)$ se trata de $\phi \geq 0$, cuando $p[i] \; = \; p[i \; - \; 1]$ ó $q[i] \; = \; q[i \; - \; 1]$, siendo $\phi \; = \; 0$, en cualquier otro caso.

ϕ es una constante con la función de disuadir todo movimiento que no sea diagonal.

Una vez que p y q han sido calculadas, el archivo MIDI original puede ser ajustado, por lo que, eventos que ocurren en $t_X[p[i]]$ son desplazados o movidos hasta $t_Y[q[i]]$, donde:

$$t_X \in \mathbb{R}^M, t_Y \in \mathbb{R}^N \quad (95)$$

Se corresponden con los tiempos asociados a los vectores de características *X e Y*, respectivamente.

DTW a menudo está restringido de modo que p y q abarcan la totalidad de X e Y, por ejemplo:

$$p[1] = q[1] = 1 \quad (96)$$

Y, $$p[L] = M\ ;\ q[L] = N \quad (97)$$

Sin embargo, los sistemas de alineamiento audio-MIDI, normalmente, permiten la coincidencia de subsecuencias (por ejemplo, el caso de un archivo MIDI que cubre solo una porción de una canción), donde en su lugar, solo se requiere que ó:

$$gM \leq p[L] \leq M \text{ ó } gN \leq q[L] \leq N \quad (98)$$

Siendo, $g \epsilon [0,1]$ **(el "gully")**

Es un parámetro que determina la proporción de la subsecuencia que debe ser correctamente emparejada. En algunos casos no es necesario su uso, lo que equivale a establecer $g = 1$. Un valor de g que es cercano a 1 permitirá cierta tolerancia a la posibilidad de que el principio o el final de la transcripción MIDI sea incorrecto.

Para el diseño de un sistema de alineamiento basado en DTW se incluye, también, la configuración de varios parámetros principales:

- **Representación de entidades (X e Y)**: previo al alineamiento, los datos de audio y MIDI deben ser convertidos a un tipo de representación intermedia, donde su distancia pueda ser computada. Esto se realiza normalmente sintetizando el archivo MIDI para obtener una señal de audio y, por tanto, computar la transformada espectral del audio grabado y el MIDI convertido.
  Para ello, existen diversos algoritmos y procesos que realizan esta función. Por ejemplo, con vectores Chroma, los cuales representan la cantidad de energía que contiene cada semitono, es una forma de representación bastante común. También, ocasionalmente se usan representaciones de datos de la forma log-magnitud con el propósito de imitar la percepción humana. En el caso de este proyecto, se ha usado una representación obtenida a partir de la conocida ya, transformada "constant Q" (CQT). Esta lo que hace es representar la cantidad de energía, al igual que con Chroma, pero de manera logarítmica y espaciada de una manera determinada con se explicó anteriormente.

- **Escalado temporal (t_x y t_y)**: Los vectores de características se calculan con frecuencia en tramas de audio cortas y superpuestas. Hay que tener en cuenta que el espaciado entre estos vectores debe ser lo suficientemente pequeño en comparación con la resolución temporal auditiva, por ejemplo, decenas de milisegundos. Ocasionalmente, se utilizan vectores de características sincrónicas de ritmo, lo que puede reducir el tiempo de cálculo y puede producir alineaciones precisas siempre que el seguimiento de ritmos sea correcto.

- **Normalización:** es de tener bien en cuenta que, estandarizar las secuencias de entrada puede ser fundamental para la representación y procesamiento de aplicaciones de datos con DTW.
  Se pueden aplicar varios tipos de normalizaciones a los vectores X e Y antes de computar su distancia local. Un ejemplo común, puede ser normalizar cada vector con normalización L^2, que se trata de un tipo de normalización basada en darle importancia a los datos que son más atípicos en el problema que se realice, y, puede ser equivalente a usar "la distancia coseno".

- **"Penalty" (ϕ):** En muchas aplicaciones de alineamiento audio-MIDI basado en DTW, no es necesario el uso de este parámetro, lo que corresponde con establecer $\phi = 0$.
  Sin embargo, siempre y cuando datos MIDI y audio tengan un "tempo" consistente, los movimientos no diagonales deben ser eliminados. Este parámetro puede ser crucial en algunos casos donde se quiere asegurar que se aprovechan las subsecuencias al completo.

Para la implementación de esta técnica de procesado de audio en Python se ha aplicado una librería de código libre llamada "djitw" que incluye todos los módulos y documentación necesarios para el uso correcto de DTW.

3.2.1 Pretty-midi

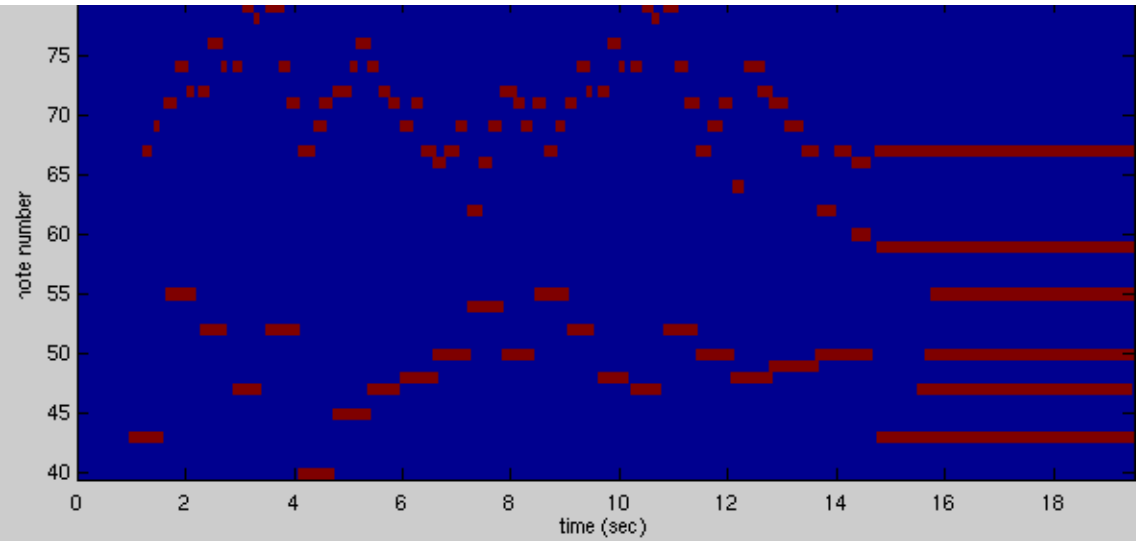
En el caso propuesto en este proyecto, para la representación de los datos y el cálculo de la matriz de distancias se ha tenido en cuenta que al tener ya una de las secuencias en formato MIDI, no era necesario el paso de procesar este MIDI en audio para su posterior procesamiento espectral, si no, que mediante el uso de una librería muy original y útil de Python, se ha trabajado a partir de los propios archivos MIDI, es decir, el archivo original MIDI de la pieza y el correspondiente MIDI del archivo de audio grabado. Esta librería es conocida como "pretty-midi".

"Pretty-midi" contiene una gran variedad de funciones/clases para el manejo y procesado de datos en formato MIDI, ya que, se trata de un formato bastante sencillo de modificar y extraer información.

Para el cálculo de la matriz de distancia se ha usado una función de "pretty-midi" que transforma el archivo MIDI, ya sea el original o el obtenido a partir de la predicción de la grabación de audio, en un tipo de representación matricial que se conoce como "piano roll".

Esto es similar, por ejemplo, a la representación utilizada en diversas aplicaciones actuales destinadas a tutoriales de piano, que hacen que aparezcan en activo las líneas que se están interpretando en tiempo real (cada línea de la matriz tiene asociada una nota del piano), dejando todas las demás inactivas y manteniéndolas activas a no activas durante el periodo de tiempo real del audio.

Ilustración xxxvi. Representación gráfica de un piano roll



Por tanto, se aplica este proceso de transformación de MIDI a "piano roll" con las dos secuencias con las que se trabaja, obteniendo dos matrices formadas por 88 filas, correspondientes cada una de ellas con las notas de un piano con $7^{1/4}$ octavas y tantas columnas como tramas temporales posean. Estas tramas seguramente no coincidirán en ambas secuencias, de ahí este proceso de alineamiento.

3.3 SISTEMA DE ALINEAMIENTO PROPUESTO

3.3.1 BASE DE DATOS DE ALINEAMIENTO

Para la evaluación del sistema de alineamiento con DTW propuesto se ha creado una pequeña base de datos, a partir de una de las piezas de audio, perteneciente al conjunto MUS de la base de datos de MAPS, en la cual mediante un cambiador de velocidad de audio online se han obtenido tres piezas derivadas de esta, pero consiguiendo tres pistas distintas, cada una con una duración propia. Para ello, se altera el tiempo de la pista original variando el coeficiente de velocidad, ya sea, acelerándola o ralentizándola.

El coeficiente de velocidad toma 3 valores distintos para realizar la evaluación de 3 pistas con tiempos distintos con respecto a la original.

Por tanto, el tiempo de cada pista nueva (t_v), será resultado de la división entre el tiempo original (t_{ori}) y el coeficiente de velocidad (v_{coef}):

$$t_v = \frac{t_{ori}}{v_{coef}} \quad (99)$$

Por lo que, si:

- $v_{coef} = 0.5$ → La pista resultante tendrá el doble de duración que la original.
- $v_{coef} = 1.5$ → La pista resultante tendrá 2/3 de la duración original.
- $v_{coef} = 2$ → La pista resultante tendrá la mitad de duración que la original.

Una vez que se tienen las pistas preparadas, se introducen cada una de ellas por el clasificador binario múltiple basado en redes neuronales, para predecir, con cada uno de los archivos, la matriz resultante correspondiente a las notas que se van interpretando en cada instante de tiempo.

Para una mejor representación, antes de practicar el alineamiento se pasan las predicciones por el módulo de "pretty-midi" detallado anteriormente. Con lo que, de esta forma, obtener los "piano rolls" de cada una de las predicciones.

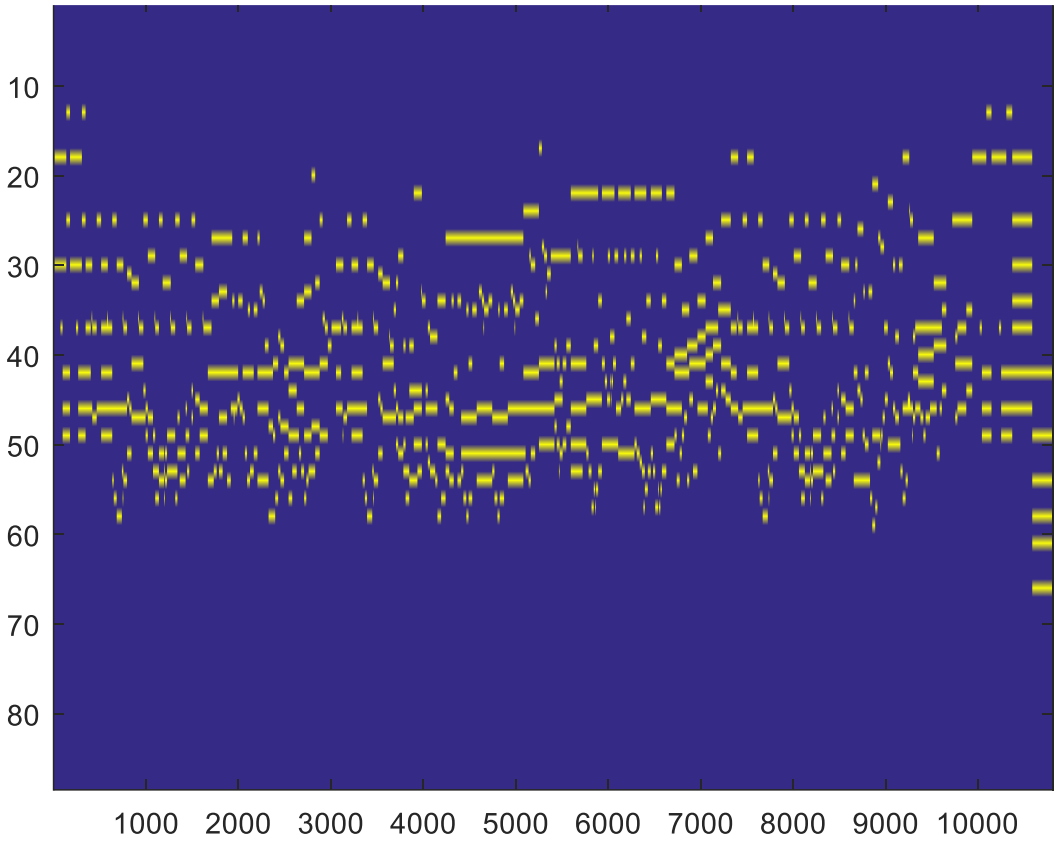
Tras esto, se realiza el proceso de alineamiento explicado, calculando la matriz de distancias entre el "piano roll" original y el correspondiente a cada pista modificada, para obtener las secuencias de tiempos asociados al emparejamiento del piano roll original con el piano roll de la predicción (p[i], q[i]).

3.3.2 EVALUACIÓN DEL SISTEMA

Una vez que ya se conoce la relación entre tramas de ambos “piano rolls”, se comprueba, a modo de evaluación del alineamiento, que esta relación coincide con el coeficiente de velocidad aplicado previamente.

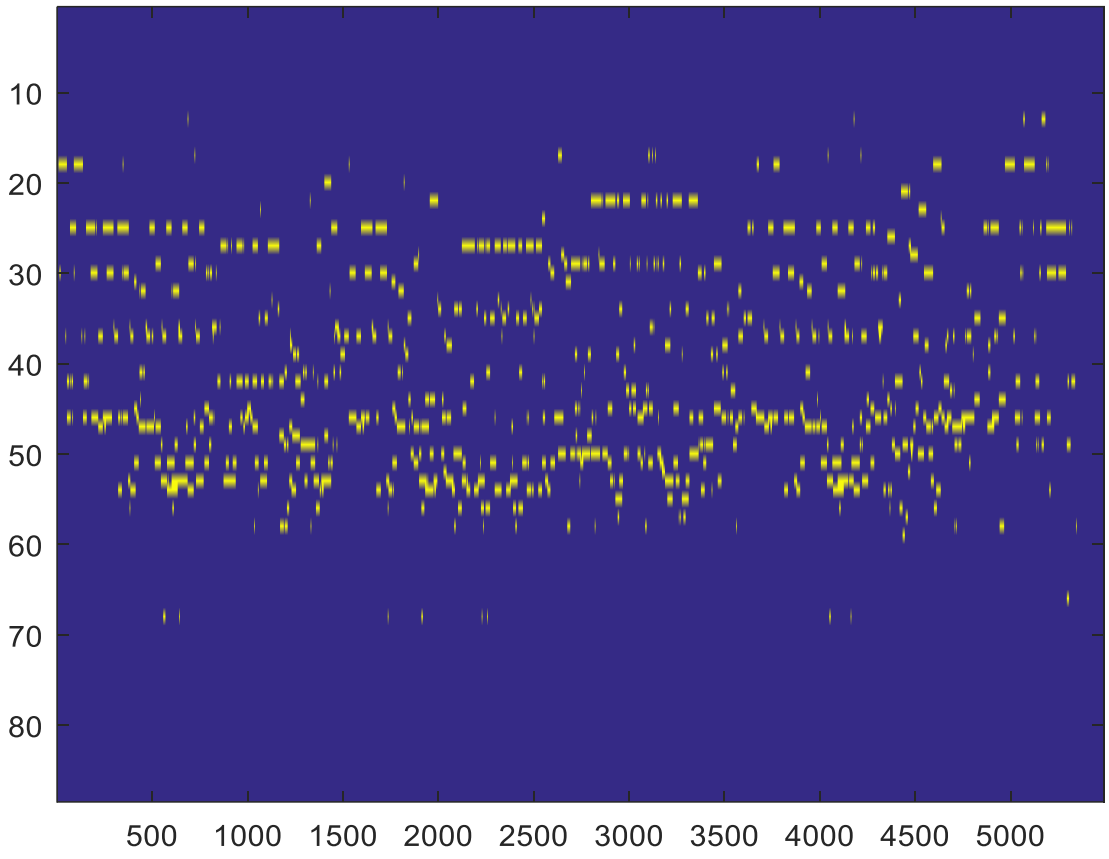
Se muestra el piano roll de la partitura original con dimensiones 88 x 10939:

Ilustración xxxvii. Piano roll de la partitura original



Caso 1: $v_{coef} = 2$

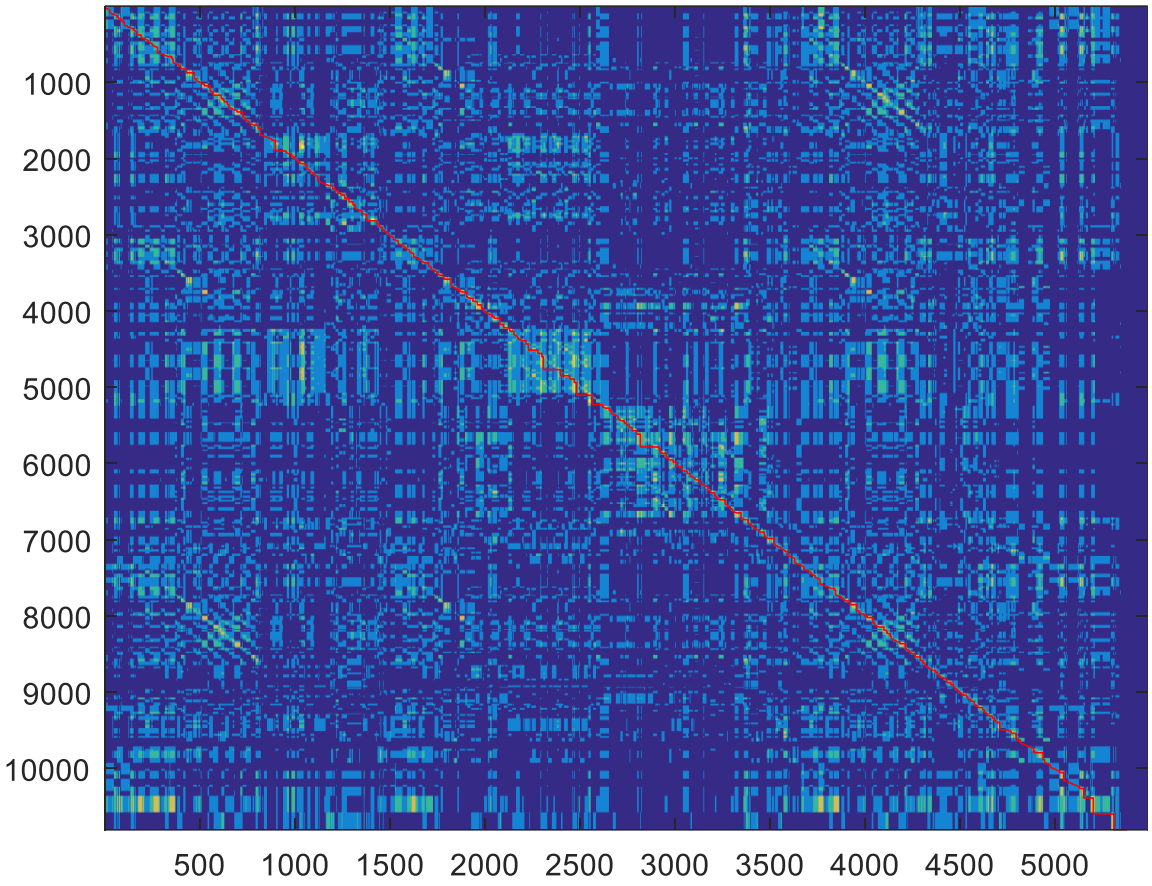
Ilustración xxxviii. Piano roll estimado con $v_{coef} = 2$



Como se puede comprobar en la figura de arriba, donde se muestra el piano roll de la estimación, en el que, debido al coeficiente de velocidad aplicado, las tramas se ven afectadas siendo reducidas a la mitad, con respecto al piano roll de la partitura original.

Representando la matriz de correlación que muestra la similitud entre ambas señales, con dimensiones 10939 x 5426. Y superponiendo en rojo las secuencias de alineamiento “p” y “q”:

Ilustración xxxix. Representación de la matriz de correlación y secuencias de alineamiento para el caso 1



Como se puede observar a simple vista, se aprecia que los valores de las secuencias de alineamiento están relacionados también por el coeficiente de velocidad establecido, por ejemplo, estando uno de los puntos de máxima similitud en la trama 1000 de la señal original (eje de ordenadas), con la trama 500 de la señal estimada (eje de abscisas), existiendo una relación constante en todo el camino de mayor parecido, igual a v_{coef}, es decir, 2.

Mostrando valores de la misma posición de las secuencias “p” y “q” se puede observar como aproximadamente se tienen los resultados adecuados.

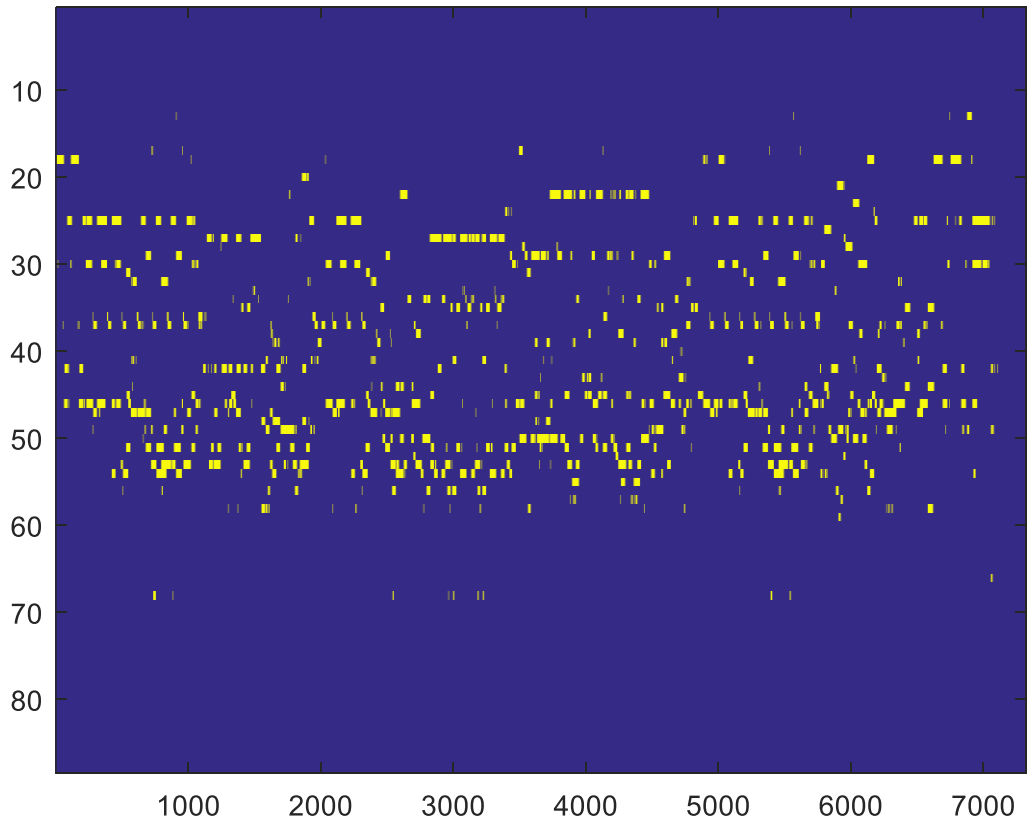
	0	1
1121	998	508
1122	998	509
1123	999	509
1124	1000	509
1125	1001	509
1126	1002	509
1127	1003	509

Tabla iii. Relación de tramas de las secuencias de alineamiento para el caso 1

En la columna 0 se representa la secuencia "p", mientras que la columna 1 se trata de la secuencia "q", correspondiéndose la trama 1000 con la trama 509, respectivamente.

Caso 2: $v_{coef} = 1.5$

Ilustración xl. Piano roll estimado con $v_{coef} = 1.5$

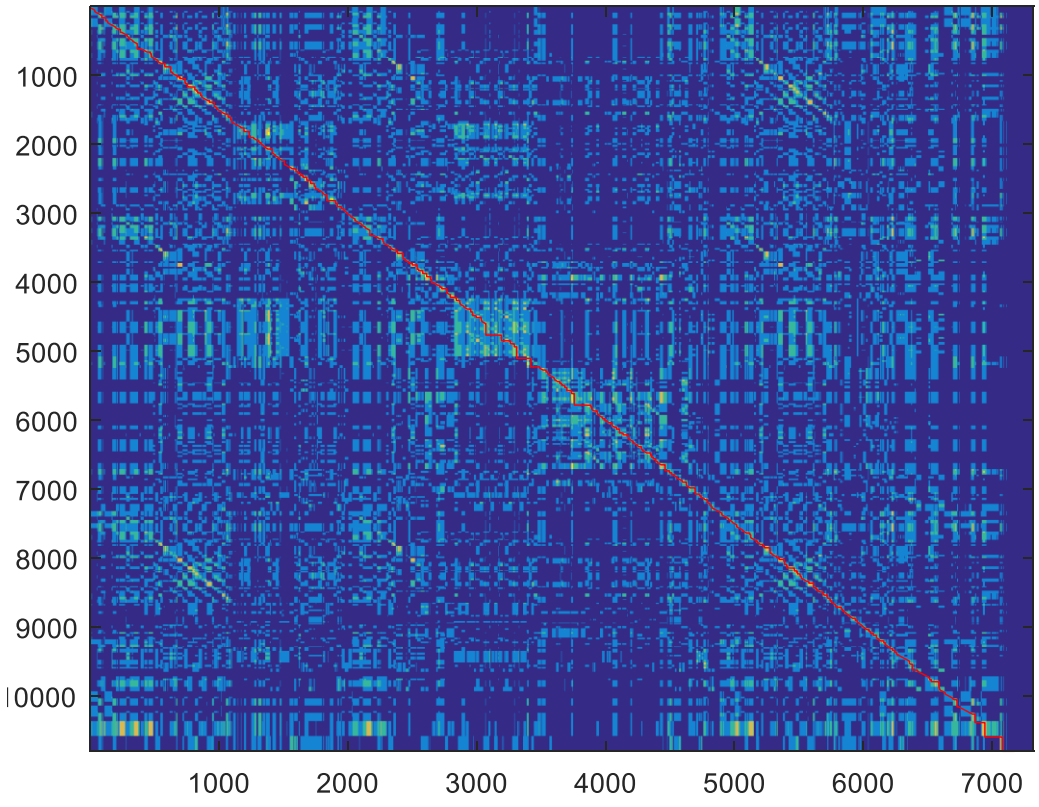


En este caso, la figura de arriba se corresponde con el piano roll de la señal estimada para alineamiento con un coeficiente de velocidad, $v_{coef} = 1.5$. Por tanto, la dimensión también se verá reducida en el eje correspondiente a las tramas que se suceden en la señal con respecto a las del MIDI original, siendo esta de 88 x 7236.

Operando de igual forma que anteriormente, se calcula la matriz de correlación realizando el producto matricial entre el piano roll original y el piano roll estimado de este caso. Por lo que, se obtiene la matriz que indica el camino de alineamiento más óptimo con una dimensión de 10939 x 7236 (nº tramas MIDI original x nº tramas piano roll estimado).

También se resaltarán los vectores de "p" y "q" obtenidos en el proceso de DTW en la figura, coincidiendo prácticamente con los máximos que presenta la matriz de correlación.

Ilustración xli. Representación de la matriz de correlación y las secuencias de alineamiento para el caso 2



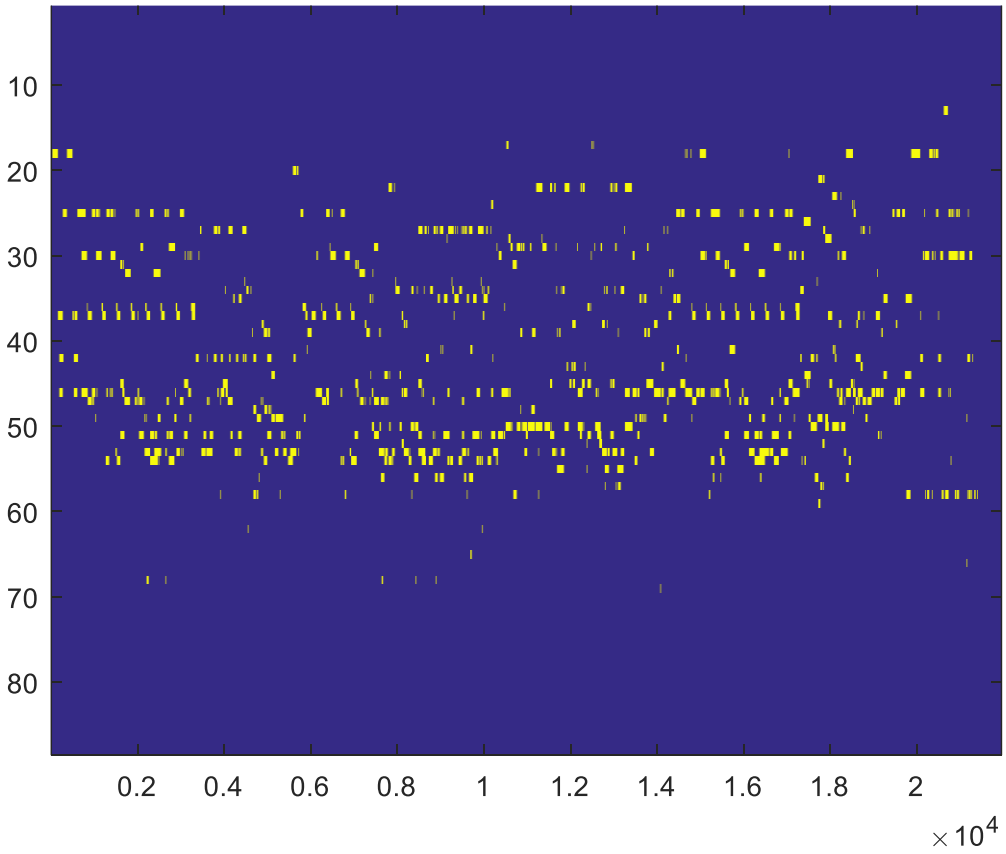
De igual forma que en el caso previo, para comprobar que efectivamente se ha realizado correctamente el alineamiento entre ambas señales o matrices, se escoge un punto del camino de máxima similitud, por ejemplo, la trama número 2000 del eje de ordenadas debe corresponder con una trama 1.5 veces menor en eje de abscisas, por lo que como se puede observar en la representación de las secuencias de alineamiento de abajo, siendo "p" la columna 0 y "q" la columna 1, esta trama 2000 correspondiente a la secuencia "p", será la trama número 1331 en la secuencia "q", que muy aproximadamente sigue la relación propuesta.

	0	1
2612	1996	1328
2613	1997	1328
2614	1998	1329
2615	1999	1330
2616	2000	1331
2617	2001	1332
2618	2002	1333
2619	2003	1334
2620	2004	1335

Tabla iv. Relación de tramas entre las secuencias de alineamiento para el caso 2

Caso 3: $v_{coef} = 0.5$

Ilustración xlii. Piano roll estimado con $v_{coef} = 0.5$



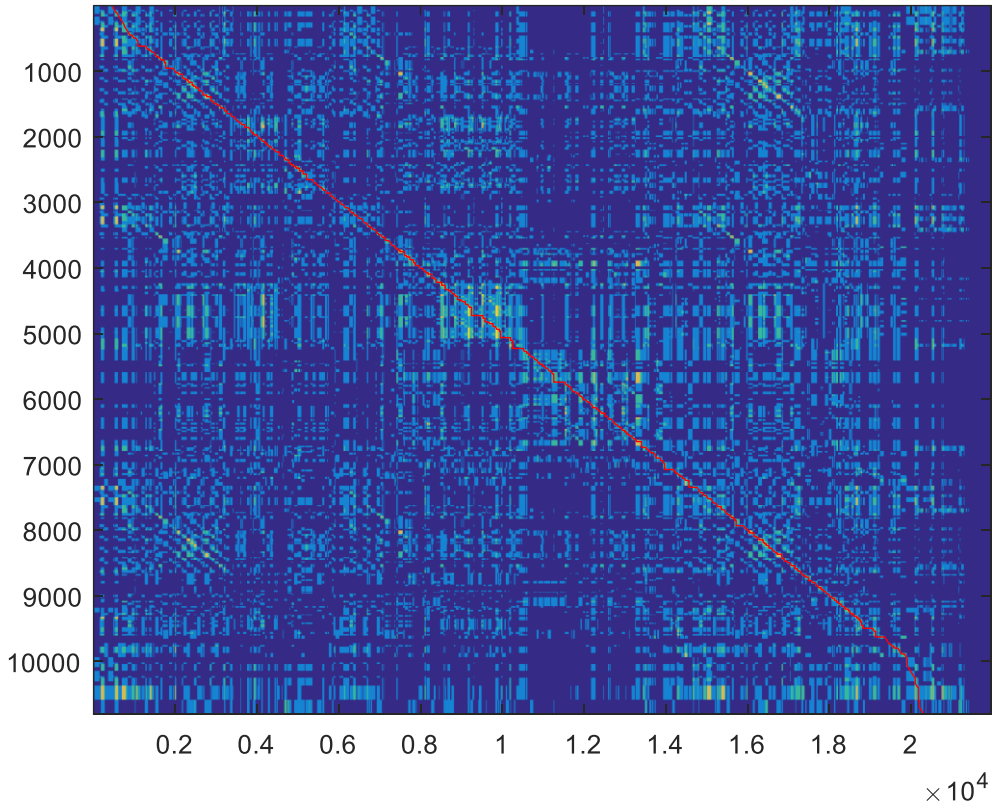
En este último caso de evaluación del sistema de alineamiento, se ha escogido un valor del coeficiente de velocidad $v_{coef} = 0.5$, por lo que, el piano roll de la predicción resultante va

a tener el doble de duración que el original. En concreto la matriz de estimación en este caso tiene dimensiones de 88 x 21878.

Para obtener las secuencias de alineamiento necesarias, esta vez se ha obtenido una matriz de correlación con dimensiones 10939 x 21878.

Como se puede observar en la gráfica siguiente, puede existir un pequeño error en el comienzo y final del alineamiento, debido a que en este caso la diferencia de duración entre ambas señales es bastante mayor y, por tanto, el alineamiento se hace más costoso.

Ilustración xliii. Representación de la matriz de correlación y las secuencias de alineamiento para el caso 3



Comprobando también en este último caso de evaluación del sistema de alineamiento, se puede afirmar cómo de bien se cumple la relación de alineamiento, escogiendo, por ejemplo, la trama 5000 de la secuencia “p” (columna 0) que corresponderá con el valor de trama número 9955 de la secuencia “q” (columna 1), siendo estos valores con la misma posición en sus respectivos vectores.

	0	1
11701	4997	9955
11702	4998	9955
11703	4999	9955
11704	5000	9955
11705	5001	9955
11706	5002	9955
11707	5003	9955

Tabla v. Relación en tramas entre las secuencias de alineamiento para el caso 3

Para todos los casos diversos de evaluación propuestos se ha configurado el sistema de alineamiento con DTW con unos parámetros concretos establecidos para su mejoría:

- "El gully" → Se ha tomado un valor de 0.98, ya que, con valores cercanos a 1 se mejora la respuesta del alineamiento, en tramas al principio y al final del proceso.
- "Penalty" → Se ha establecido este parámetro en cada caso, tomando la media resultante de la matriz de correlación siendo este un valor entre [0-1], con lo que obtener mejores resultados en alineamientos con pistas de "tempo" considerable.

4. SISTEMA FINAL PROPUESTO

4.1 INTRODUCCIÓN

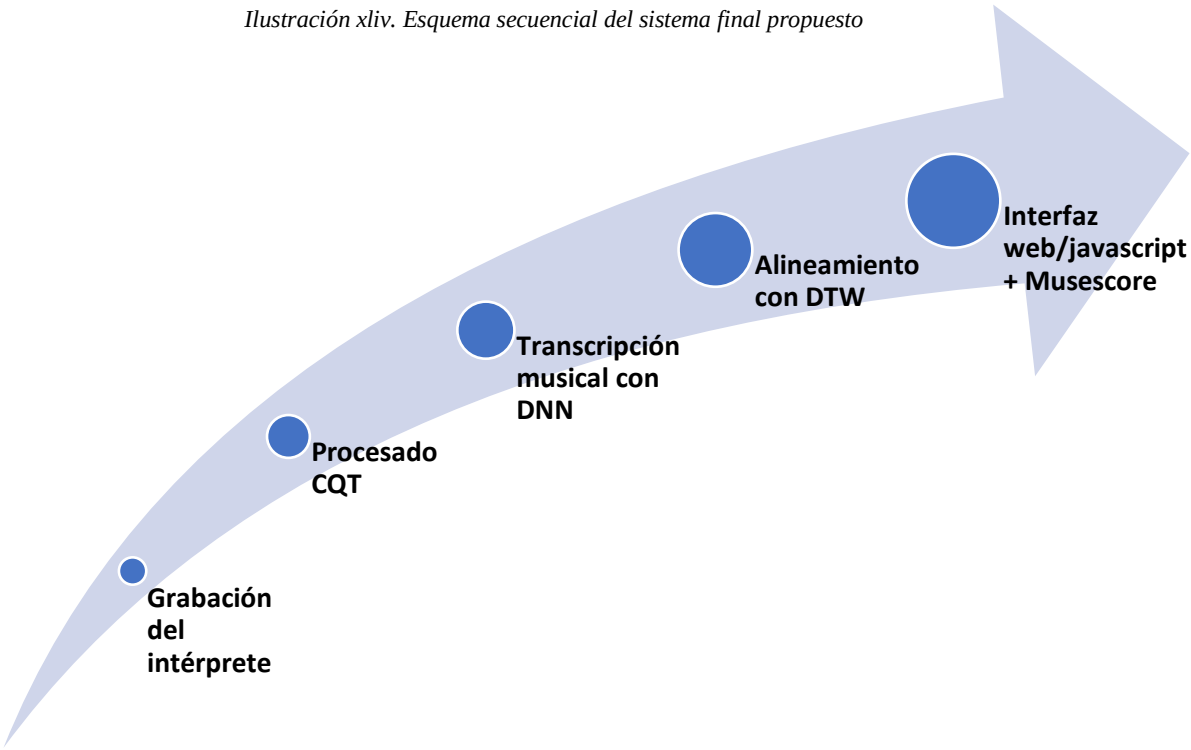
Una vez que ya se ha detallado de manera independiente cada uno de los apartados que posee este proyecto, dando cuenta de en qué consiste cada parte de manera teórica e implementando un sistema de evaluación de cada sección para probar su funcionalidad antes de pasar a sintetizar todo el sistema en uno. Esto será lo que se exponga en este apartado, hacer un seguimiento de todo lo explicado y unirlo en una aplicación práctica determinada.

Por tanto, la aplicación propuesta básicamente consistirá en un sistema web de navegación de audio, donde, a partir de la partitura MIDI de una pieza concreta cargada en un software libre de composición musical, el sistema se va a ser capaz, desde un navegador web, de sincronizar cualquier grabación de dicha pieza interpretada con la partitura original mediante un tipo de comunicación llamada, OSC.

Para que cualquier grabación de interpretación quede bien ajustada en el tiempo y el software pueda hacer un seguimiento del compás por el que transcurre el audio en tiempo real, aquí será donde entre en juego el sistema de transcripción musical basado en redes neuronales y su posterior alineamiento con DTW.

Para una mejor comprensión del modelo del sistema de aplicación final se presenta el siguiente esquema secuencial que muestra cada una de las secciones seguidas para la realización total del proyecto:

Ilustración xliv. Esquema secuencial del sistema final propuesto



A continuación, en los siguientes subapartados se explica cada una de las secciones seguidas en el proceso de realización del caso práctico.

4.2 BASE DE DATOS PROPUESTA

El primer paso, está claro que, va a ser la selección de un conjunto de piezas de música clásica, concretamente de piano. Se ha descargado una grabación de cada una de las piezas elegidas de un intérprete aleatorio (cada una con un tempo, fraseo o velocidad a gusto del pianista), desde cualquier programa o soporte de audio y/o video, en este caso se ha usado YouTube para obtener las interpretaciones correspondientes.

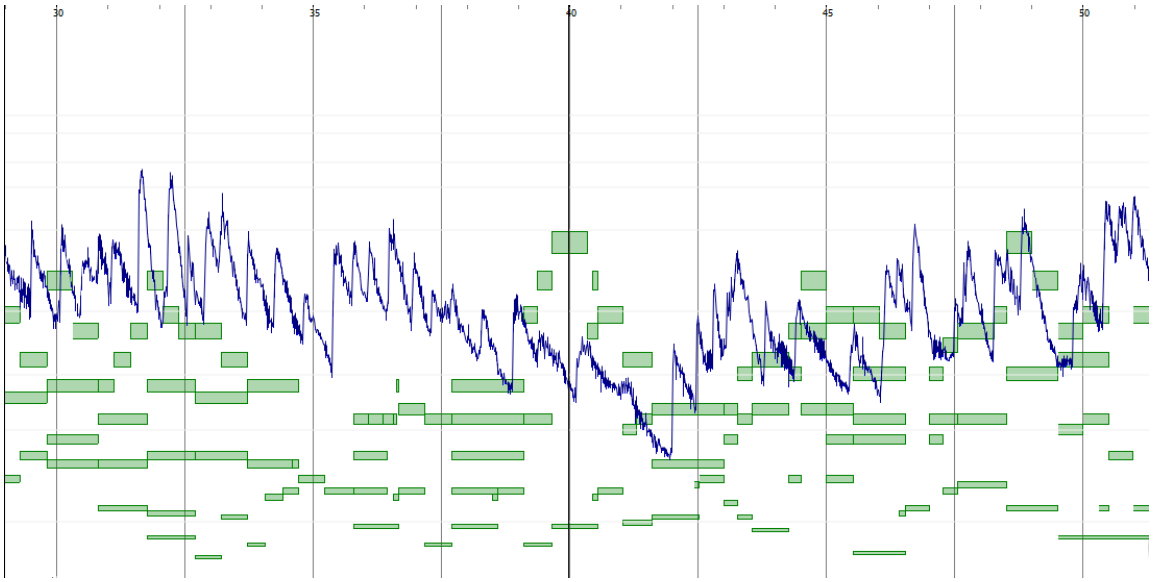
Todas las reproducciones van a tener una frecuencia de muestreo (fs) de 44100 Hz. Independientemente, no es importante si estas son mono canal o estéreo, ya que más adelante se convertirán a "mono", todas y cada una de ellas, para su procesado.

Mediante la base de datos de música MAPS, se extraerá el correspondiente archivo MIDI de las piezas escogidas, aunque también puede ser descargado de internet o cualquier otro sitio si fuese posible.

Para hacer la demostración práctica de como sucede el sistema y hacer prueba de que funciona, se ha escogido, al azar, una pista de audio correspondiente a una grabación interpretada y importada a YouTube de una pieza de Isaac Albéniz (España Op.165, 2º movimiento, Tango), la cual se ha obtenido de MAPS la correspondiente partitura en formato MIDI. La señal temporal de la grabación contiene datos en una duración de 2 minutos y 49 segundos, aproximadamente, con una frecuencia de muestreo de 44100Hz.

Mediante un software de edición de audio, por ejemplo, Sonic Visualiser, se puede comprobar que efectivamente la señal de entrada grabada por el intérprete no está alineada con el archivo MIDI de la partitura original, ya que, este será el objetivo que tendrá que obtener el sistema diseñado. Se muestra un ejemplo en la siguiente figura:

Ilustración xlv. Demostración de NO alineamiento con Sonic Visualiser



Como se puede apreciar a simple vista, los máximos de los flancos de la señal temporal no coinciden con el comienzo de cada nota MIDI, si no que están desplazados.

4.3 PROCESADO CON CONSTANT-Q TRANSFORM

Una vez que se tiene ya una señal temporal de entrada preparada, se procede a realizar el correspondiente procesamiento de transformación "tiempo-frecuencia". Para ello, como se explicó anteriormente se va a usar la conocida como, transformada "Constant-Q".

Como ya se ha detallado hay que establecer 5 parámetros fundamentales para realizar esta transformación:

1. La señal temporal de entrada con la que se va a trabajar. En este caso, la grabación de interpretación elegida previamente.

2. La frecuencia de muestreo de la señal temporal que va a ser, **fs** = 44100 Hz.

3. **Hop_length**: la longitud de muestras de tiempo que corresponderán a una trama en frecuencia. En este caso, "Hop_length" = 512, es decir, de cada 512 muestras de la señal temporal habrá 1 trama en la señal con dominio espectral.

4. **Bins_per_octave**: se trata al número de filtros usados por octava para su representación en frecuencia. En el proyecto se ha establecido un valor de 36 filtros por octava.

5. **N_bins**: este parámetro consiste en el número total de "bins" obtenidos por cada una de las octavas, es decir, 7 octavas teniendo cada una 36 filtros, hacen un total de 252 coeficientes.

Para demostrar cómo quedaría la matriz de coeficientes transformados, aplicando estas características de la CQT, sabiendo que la duración de la señal es igual a 169s, a una fs = 44100 Hz se obtiene que la señal de entrada tendrá:

$$L_s = L_t \cdot f_s = 169s \cdot 44100Hz = 7.452.900\ muestras \qquad (100)$$

Siendo, L_s la duración en muestras y L_t la duración en segundos.

Por tanto, como la dimensión de la matriz de entrada al clasificador binario va a estar formada por 252 coeficientes, correspondientes a todos los filtros usados en la transformación, en un eje, teniendo en el otro el número de tramas con la que se forma la señal. Sabiendo que 1 trama se obtiene a partir de cada 512 muestras (valor establecido en el parámetro "Hop_length"):

$$L_f = \frac{L_s}{Hop\_length} = \frac{7.452.900}{512} \approx 14557\ tramas \qquad (101)$$

Por lo tanto, a la salida del proceso de transformación se obtendrá una señal en el dominio espectral, correspondiente con una matriz, $X_{i,j}$:

Siendo, $$i = (0, 1, \ldots, 251) \quad (102)$$

$$j = (0, 1, \ldots, L_f - 1) \quad (103)$$

Es decir, la dimensión de la matriz será de $252 \times L_f$.

4.4 TRANSCRIPCIÓN MUSICAL CON DNN

Una vez que ya se tiene la señal de entrada lista en el dominio de la frecuencia se procede a introducirla en el predictor. Antes de este paso, deberá de normalizarse la matriz para obtener mejor resultado en la resolución final, ya que así, los valores de los coeficientes de la matriz se encontrarán en un rango adecuado para que la red sea capaz de predecirlos con un rango de valores similar al que ésta ha sido entrenada.

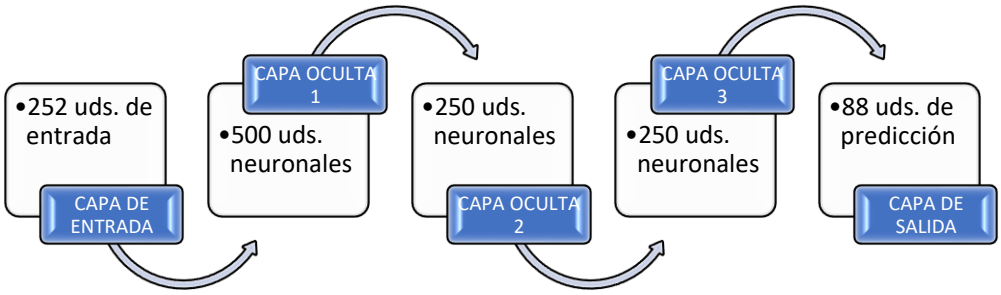
Por esto, la normalización se realizará de la manera explicada en el apartado 2 de transcripción musical donde se menciona el tratamiento de los datos antes del proceso predictivo, en el cual, se aplicará el método común de normalización a partir de una media y desviación típica, calculada conjuntamente entre los valores que se obtengan con los datos con los que se entrenó y validó la red y estos parámetros estadísticos correspondientes a la matriz que va a ser predicha.

El siguiente paso del procedimiento va a ser el proceso de predicción musical basada en una red neuronal profunda.

Para ello, se ha escogido la configuración de red con la que se obtuvieron los mejores resultados en el proceso de evaluación del apartado 2.5.

A modo de breve recordatorio, la red más óptima se correspondía con un tipo de red neuronal profunda basada en "perceptrones multicapa". Formada por 5 capas:

Ilustración xlvi. Diagrama secuencial de la red final



En las que, como se puede observar, la primera capa corresponde con la entrada, después le siguen 3 capas ocultas y finalmente la capa de salida que es la encargada de mostrar la matriz resultante del clasificador y por tanto, la predicción del mismo.

Las tres capas ocultas están constituidas por funciones de activación ReLU y la capa externa posee neuronas con activación sigmoide, típicas en problemas de clasificación binaria.

Por otro lado, la función de coste con la que se calcula la predicción está basada en la función de entropía cruzada binaria, detallada en el apartado 2.4.

También, es necesario resaltar el uso de un optimizador de tipo ADAM y, además, para evitar sobre entrenamiento, a modo de regulación se ha implementado en cada capa un valor de "Dropout" igual a 0.3.

Por tanto, aplicando todo esto y realizando un proceso de propagación hacia delante con los datos de entrada a predecir, a través de la red propuesta, una vez ya entrenada, se obtiene una matriz resultante que se corresponde con la predicción exportada del sistema.

Esta matriz de salida tendrá un dimensionamiento diferente al de entrada, ya que, fijándose en la arquitectura de la red propuesta anteriormente, la capa de salida formada por 88 unidades va a ir mostrando, por tanto, 88 valores resultantes de cada trama introducida al sistema compuesta por 252 coeficientes espectrales.

Con lo cual, se dispondrá de una matriz de predicción $Y_{k,j}$, donde:

$$k = (0, 1, \ldots, 87) \quad \textbf{(104)}$$

$$j = (0, 1, \ldots, L_f - 1) \quad \textbf{(105)}$$

Para ampliar, no está de más aclarar que, los 88 valores obtenidos por la predicción de cada trama en el predictor no se corresponden directamente con valores en forma binaria, si no, un valor de la probabilidad, con la que el sistema ha estimado que la predicción se corresponda con esa unidad, estando cada una de las unidades asociadas a una nota en formato MIDI, por tanto, indirectamente prediciendo la probabilidad de que aquella, o aquellas, con mayor índice probable sean las que estén siendo interpretadas en ese instante de tiempo.

Finalmente, por tanto, en esta parte del proceso se reforma esta matriz resultante activando en 1 aquellos valores de probabilidad que se encuentren por encima de un valor umbral establecido y por consiguiente dejando a 0 todo lo demás, obteniendo la matriz de predicción binaria con la que se procederá, a continuación, al proceso de alineamiento.

4.5 ALINEAMIENTO CON DTW

El último paso, antes de proceder a unir todo este método de procesamiento y predicción de audio con la interfaz web con la que visualizar cómo funciona el sistema, va a ser aplicar la técnica de alineamiento basada en DTW, expuesta en el apartado 3.

Esta herramienta calcula, a partir del piano roll del MIDI original y la matriz binaria resultante en la predicción (esta matriz también se corresponde con el piano roll estimado de la grabación de interpretación), dos secuencias, llamadas "p y q", detalladas en el apartado 3.3, las cuales, mediante el cálculo de una matriz de correlación entre la matriz del MIDI original y la matriz de predicción, determinan el camino de máxima similitud entre ambas matrices con respecto al tiempo.

Sabiendo que los valores que contienen ambas secuencias se corresponden con valores de determinadas tramas que se suceden a lo largo de las matrices, resulta que el proceso de alineamiento hace que se emparejen los "piano rolls" en el tiempo, de modo que los valores de la misma posición de p y q corresponderán con los instantes de las señales (tanto la original, como de estimación) en tramas en los que éstas deben coincidir o tener mayor grado de similitud en el proceso de correlación, es decir, de forma más simple, la señal original, en los valores concretos de tramas que se suceden en las posiciones de la secuencia "p", tendrá que coincidir con los instantes de la señal estimada en el predictor que indican los valores de las tramas correspondientes a la secuencia "q".

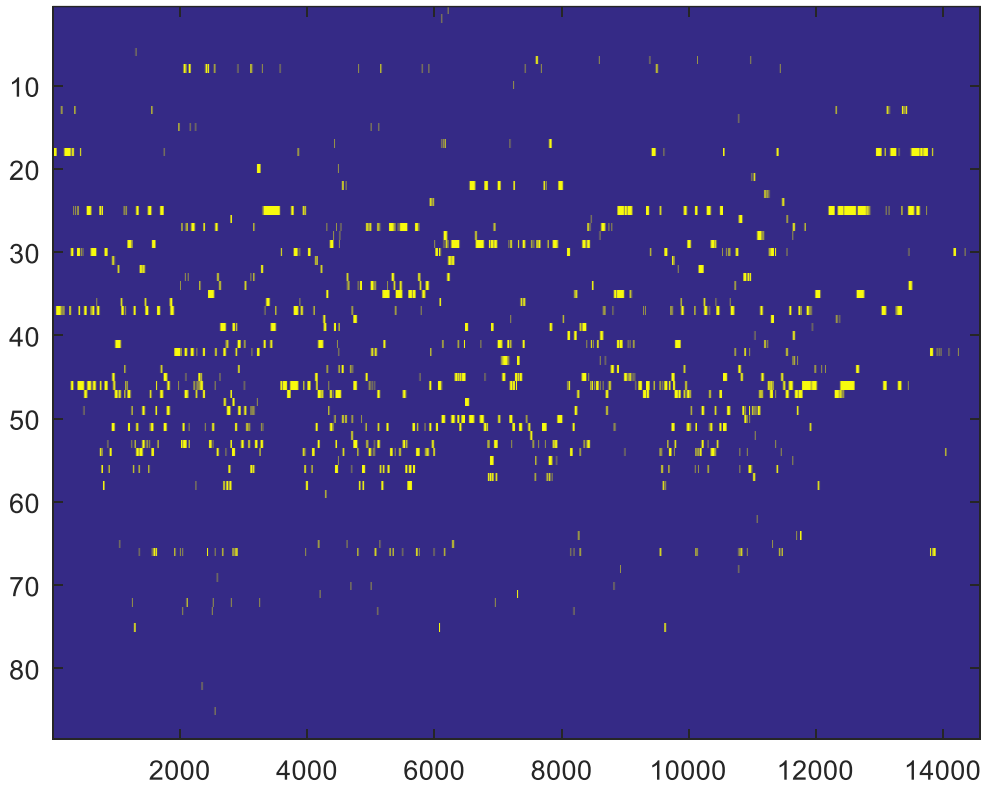
Por ejemplo, si p[100] = 50 y q[100] = 80, se puede deducir que la señal original en la trama número 50 tiene que coincidir espectralmente con la señal estimada en la trama número 80.

Por tanto, teniendo a punto el piano roll original y el piano roll estimado se procede a calcular mediante una librería de Python llamada "djitw" que posee un módulo para DTW, como ya se mencionó en el apartado 3.2, las secuencias de tiempos "p y q" asociadas a estos "piano rolls".

Sabiendo que:

- la matriz del piano roll original tiene una dimensión de 88 x 10939
- la matriz de piano roll de estimación posee una dimensión de 88 x 14557

Ilustración xlvii. Piano roll estimado del sistema propuesto final

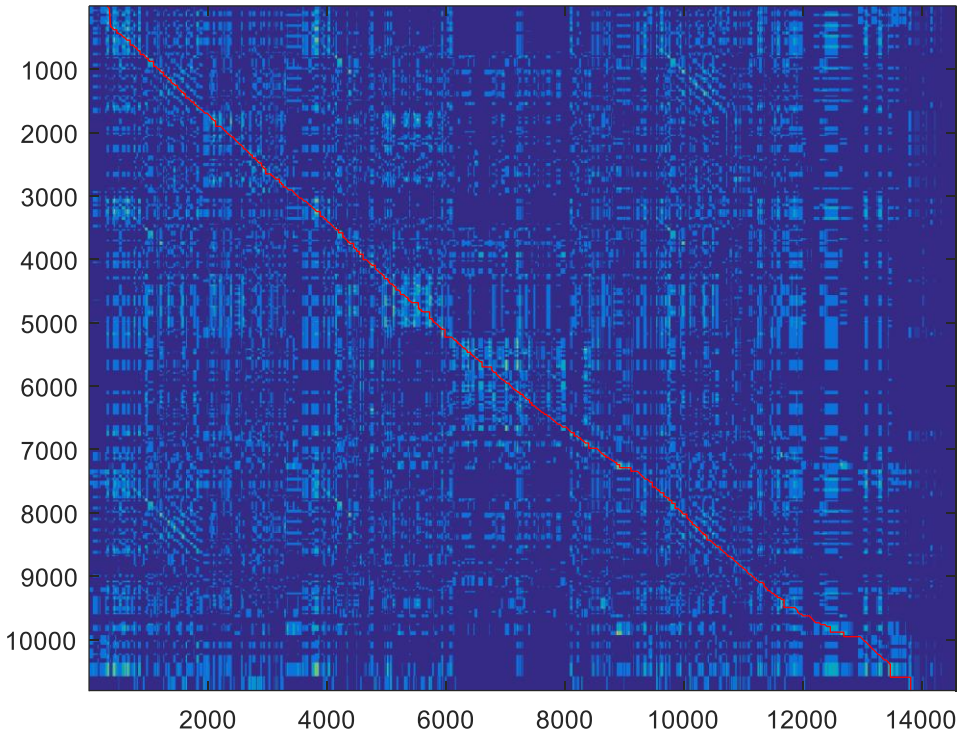


También, estableciendo los mismos parámetros usados en la evaluación del sistema de alineamiento con DTW:

- "El gully" → Se ha tomado un valor de 0.98, ya que, con valores cercanos a 1 se mejora la respuesta del alineamiento, en tramas al principio y al final del proceso.
- "Penalty" → Se ha establecido este parámetro tomando la media resultante de la matriz de correlación siendo este un valor entre [0-1], ya que, puede ser una solución adecuada al alineamiento de este proyecto, debido al "tempo" agitado de la composición con la que se ha propuesto el sistema final.

Calculando mediante el proceso de minimización propuesto en el apartado 3.2, se obtiene una matriz de correlación con dimensiones 10939 x 14557:

Ilustración xlviii. Representación de la matriz de correlación y las secuencias de alineamiento para el sistema propuesto final



Donde las secuencias "p y q", marcadas en rojo en la figura anterior, ambas estarán formadas con la misma dimensión de 1 x 18584, resaltando el camino de máxima similitud entre las matrices.

Estas secuencias serán los datos introducidos en la pequeña interfaz web creada para monitorizar los resultados.

4.6 INTERFAZ WEB/JAVASCRIPT CON FLASK/PYTHON + MUSESCORE

4.6.1 Introducción

Para monitorizar los resultados obtenidos durante el proceso de transcripción + alineamiento, se ha creado una pequeña aplicación web apoyada sobre Flask con Python, que lee los datos exportados del sistema de alineamiento, los analiza en tiempo real sincronizando la señal grabada por el intérprete y la partitura original, de modo que, mediante una interpolación de las secuencias de alineamiento "p" y "q", se obtiene el compás en el que se encuentra la reproducción en cada momento. Por tanto, ese compás va siendo transmitido, a través de una comunicación entre el interfaz web y una app de notación musical (donde se encuentra la partitura original en MIDI), mediante un tipo de comunicación UDP típica entre este tipo de sistemas, conocida como OSC.

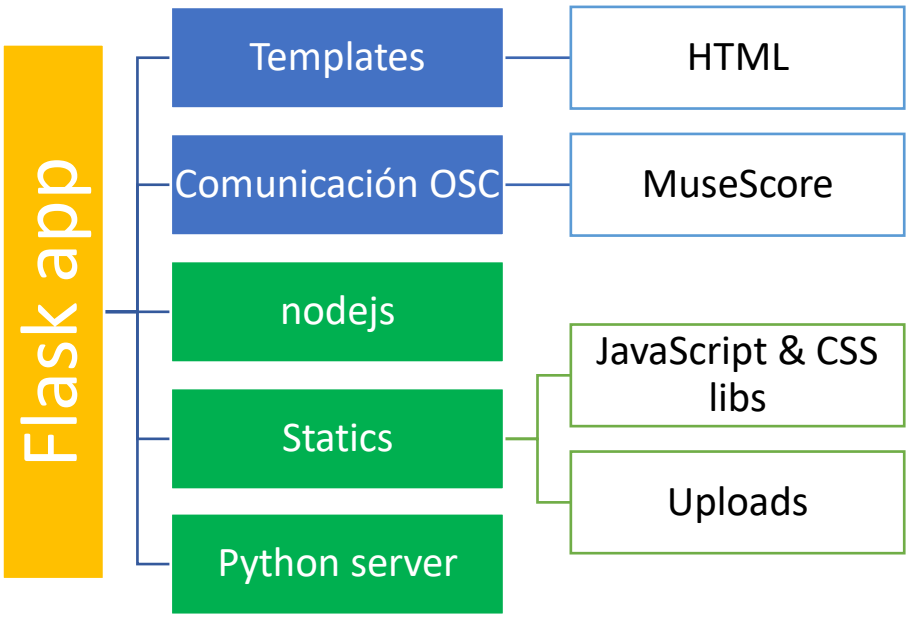
Para ello, antes de entender el sistema completo propuesto se detallan cada una de las partes diferenciadas en esta sección.

Esta aplicación de representación de los resultados va a disponer de dos partes diferenciadas en el esquema siguiente:

- ✓ En azul: el proceso a nivel de cliente de la aplicación, es decir, el interfaz web a nivel HTML junto con la comunicación de los datos y monitorización en el software de notación musical, usado MuseScore.

- ✓ En verde: corresponde con la parte interna de la aplicación web, es decir, a nivel de servidor, que se encarga de abrir los puertos pertinentes para la comunicación, llamar a las funciones necesarias en JavaScript solicitadas por el cliente y dotar al sistema de los archivos necesarios.

Ilustración xlix. Esquema sobre la Flask app



Antes de detallar más a fondo en que consiste esta estructura de desarrollo web de tipo cliente/servidor, se va a hacer un breve repaso sobre algunos conceptos básicos en relación a los lenguajes utilizados, así como el entorno de trabajo.

4.6.2 HTML

Definiéndolo de forma sencilla, "HTML es el lenguaje que se utiliza para crear las páginas web a las que se accede mediante internet". Más concretamente, HTML es el lenguaje con el que se "escriben" la mayoría de páginas web.

Aunque HTML es un lenguaje que utilizan los ordenadores y los programas de diseño de páginas web, es muy fácil de entender y escribir por parte de las personas. En realidad, HTML son las siglas de “HyperText Markup Language”.

El lenguaje HTML es un estándar reconocido en todo el mundo y cuyas normas define un organismo sin ánimo de lucro llamado “World Wide Web Consortium”, más conocido como W3C. Como se trata de un estándar reconocido por todas las empresas relacionadas con el mundo de internet, una misma página escrita en HTML se visualizará de forma muy similar en cualquier navegador bajo distintos sistemas operativos.

HTML es un lenguaje de etiquetas (también llamado lenguaje de marcas) y las páginas web habituales están formadas por cientos o miles de pares de etiquetas. De hecho, las letras "ML" de la sigla HTML significan “markup language”, que es como se denominan en inglés a los lenguajes de marcas. Además de HTML, existen muchos otros lenguajes de etiquetas como XML, SGML, entre otros.

La principal ventaja de los lenguajes de etiquetas es que son muy sencillos de leer y escribir por parte de las personas y de los sistemas electrónicos.

Brevemente, mostrando la evolución de la tecnología Web:

- Web 1.0 - Usuarios conectándose a la web y la web como servicio de información estática.

- Web 2.0 - Personas conectándose a personas, la inteligencia colectiva como centro de información y la web es sintáctica.

- Web 3.0 - Aplicaciones web conectándose a aplicaciones web, las personas siguen siendo el centro de la información y la web es semántica.

- Web 4.0 - Personas conectándose con personas y aplicaciones web de forma ubicua, se añaden tecnologías como la inteligencia artificial, la voz como vehículo de intercomunicación para formar una "Web Total".

Por tanto, llegando a la cuarta generación de la tecnología Web, se han alcanzado una serie de características y preferencias muy avanzadas, como pueden ser:

- Seguridad y privacidad, todo lo relacionado con autenticaciones, encriptación, protección de identidad y actividad en línea.

- Diseño y desarrollo de la web, en cuanto a estilo, formato, gráficos, animación y tipografía.

- Interacción con distintos equipos como sistemas de sensores y Bluetooth.

- Ciclo de uso de aplicación para administración de tareas fuera de conexión y sincronización.

- Medios y comunicaciones en tiempo real, para efectos, por ejemplo, de transmisiones en vivo (streaming).

- Desempeño y afinación de la capacidad y precisión en la respuesta y descarga de sitios web con sus funciones.

- Usabilidad y accesibilidad, para un web internacional, multilingüe y de acceso para personas con distintas discapacidades.

- Servicios como pagos y web social.

De forma paralela a su actividad con HTML, W3C ha continuado con la estandarización de XHTML, una versión avanzada de HTML y basada en XML.

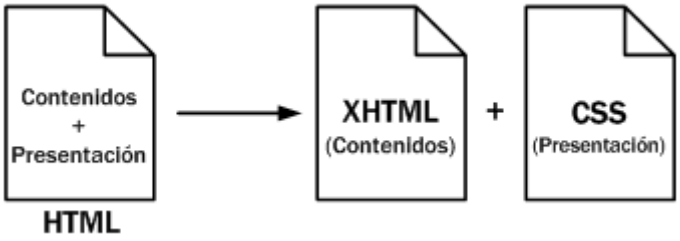
4.6.2.1 HTML + CSS

Originalmente, las páginas HTML sólo incluían información sobre sus contenidos de texto e imágenes. Con el desarrollo del estándar HTML, empezaron a incluir también información sobre el aspecto de sus contenidos: tipos de letra, colores y márgenes.

La posterior aparición de tecnologías como JavaScript, provocaron que las páginas HTML también incluyeran el código de las aplicaciones (scripts) que se utilizan para crear páginas web dinámicas.

Incluir en una misma página HTML los contenidos, el diseño y la programación complica en exceso su mantenimiento. Normalmente, los contenidos y el diseño de la página web son responsabilidad de diferentes personas, por lo que es adecuado separarlos. CSS es el mecanismo que permite separar los contenidos definidos mediante XHTML y el aspecto que deben presentar esos contenidos:

Ilustración l. Composición básica de un documento HTML



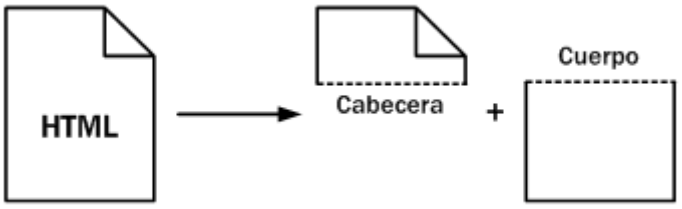
Otra ventaja de la separación de los contenidos y su presentación es que los documentos XHTML creados son más flexibles, ya que se adaptan mejor a las diferentes plataformas: pantallas de ordenador, pantallas de dispositivos móviles, impresoras, entre otros.

De esta forma, utilizando exclusivamente XHTML se crean páginas web "feas" pero correctas. Aplicando CSS, se pueden crear páginas "bonitas" a partir de las páginas XHTML correctas.

4.6.2.2 Estructura básica de un documento HTML

Las páginas HTML se dividen en dos partes: la cabecera y el cuerpo. La cabecera incluye información sobre la propia página, como por ejemplo su título y su idioma, que no se visualiza. El cuerpo de la página incluye todos sus contenidos, como párrafos de texto e imágenes.

Ilustración li. Estructura de un documento HTML



El cuerpo (llamado body en inglés) contiene todo lo que el usuario ve en su pantalla y la cabecera (llamada head en inglés) contiene todo lo que no se ve (con la única excepción del título de la página, que los navegadores muestran como título de sus ventanas).

Aunque el cuerpo contenga todo lo que se muestra en la pantalla, también suele contener todos los scripts escritos en JavaScript para dotar de dinamismo y funcionalidad a la página o aplicación Web.

4.6.3 JavaScript

4.6.3.1 Introducción

Es un lenguaje de programación que te permite crear contenido nuevo y dinámico, controlar archivos de multimedia, crear imágenes animadas y entre muchas otras.

Principalmente, utilizado en navegadores web para escribir páginas web dinámicamente, pero a menudo también del lado del servidor.

4.6.3.2 Un poco de historia

Concebido como un lenguaje de servidor por Brendan Eich (cuando trabajaba para Netscape Corp). JavaScript pronto se integró a Netscape Navigator 2.0 en septiembre de 1995. JavaScript gozó de un éxito inmediato, por lo que Microsoft introdujo en Internet Explorer 3.0 un soporte de JavaScript bajo el nombre de JScript en Agosto de 1996.

En noviembre de 1996, Netscape empieza a trabajar junto con ECMA International para convertir JavaScript en un estándar dentro de la industria. Desde entonces, la versión estandarizada de JavaScript es conocida bajo el nombre de ECMAScript y la especificación es conocida como ECMA-262.

La más conocida (y más implementada) versión de ECMAScript es ECMA-262 3ra edición. La versión actual, disponible en todos los navegadores web modernos es ECMA-262 5ta edición.

Actualmente, se está trabajando en una 6ta edición de ECMAScript.

4.6.3.3 Usos y características

JavaScript es principalmente utilizado en los navegadores web, permitiendo a los desarrolladores realizar una enorme cantidad de funcionalidades: manipular el contenido de las páginas a través del DOM, manipular datos con AJAX, crear gráficos con Canvas, interactuar con el dispositivo que ejecuta el navegador utilizando varias APIs, etc.

A modo de ampliación, se detallan unas breves definiciones de los conceptos más necesarios de comprender para entender este lenguaje.

- **API** (Interfaz de Programación de Aplicaciones) es un término común en Ciencias de la Computación. Es un término genérico que es usado para cubrir todos los requerimientos técnicos para tener varios componentes de software capaces de funcionar con los demás. En un contexto web, su uso es un poco más restrictivo, ya que es usado casi sólo con JavaScript. En ese contexto, una API es usualmente un conjunto de métodos, propiedades y eventos para con el fin lograr ciertas tareas.

- ***DOM*** (Document Object Model, en español, Modelo de Objetos del Documento) es una API definida para representar e interactuar con cualquier documento HTML o XML. El DOM es un modelo de documento que se carga en el navegador web y que representa el documento como un árbol de nodos, en donde cada nodo representa una parte del documento (puede tratarse de un elemento, una cadena de texto o un comentario).

- **AJAX** (de las siglas en inglés, Asynchronous JavaScript And XML) es una práctica de programación que combina HTML, CSS, JavaScript, el DOM y el "object XMLHttpRequest" para construir páginas web más complejas. El uso de AJAX permite actualizar partes de la página web en lugar de recargar la página entera. AJAX permite además trabajar de manera asíncrona, lo que significa que el código continúa ejecutándose mientras esa parte de tu página web está intentando recargarse (en comparación, una carga síncrona bloquearía tu código hasta que esa parte de la página web terminara de cargarse).

El reciente aumento de las APIs disponibles en los navegadores, así como una gran mejora en el rendimiento, lo convierte en uno de los lenguajes de programación más usados en el mundo.

Recientemente, JavaScript ha vuelto al servidor con el éxito de Node JS. Esta plataforma provee un entorno completo de ejecución JavaScript fuera del navegador, capaz de ser usado en cualquier plataforma (Linux, MacOS y Windows). No es el único, pero el más usado recientemente para correr JavaScript fuera de los navegadores. Esto permite el uso de JavaScript como un lenguaje de scripting para automatizar ciertos procesos, así como construir HTTP completamente funcional y servidores Web Sockets.

4.6.4 FLASK CON PYTHON

4.6.4.1 Introducción

Flask se trata de un micro web Framework escrito en Python y concebido para facilitar el desarrollo de Aplicaciones Web bajo el patrón MVC. Comúnmente, es clasificado como un microframework, ya que no requiere librerías en particular.

De principio en la instalación no se tienen todas las funcionalidades que se pueden necesitar, pero de una manera muy sencilla se pueden extender el proyecto con nuevas funcionalidades por medio de plugins.

Está basado en la especificación WSGI de Werkzeug y el motor de templates Jinja2 y tiene una licencia BSD.

4.6.4.2 ¿Qué es un Framework?

Actualmente en el desarrollo moderno de aplicaciones web se utilizan distintos Frameworks que son herramientas que nos dan un esquema de trabajo y una serie de utilidades y funciones que nos facilita y nos abstrae de la construcción de páginas web dinámicas.

En general, en el mundo de Python el Framework más conocido y usado con diferencia es Django, pero Flask puede ser una opción óptima para este tipo de aprendizajes, ya que, posibilita la creación de aplicaciones web muy complejas, pero de manera más rápida e intuituva.

Ventajas de usar un Framework.

- Proporciona una estructura del proyecto, es decir, todas las Apps que estén construidas con Flask van a tener los mismos elementos y los mismos ficheros.
- Facilita la colaboración.
- Es fácil encontrar librerías adaptadas al Framework.

4.6.5 COMUNICACIÓN OSC + MuseScore

4.6.5.1 Introducción

Open Sound Control (OSC) es un protocolo para la comunicación entre computadoras, sintetizadores de sonido y otros dispositivos multimedia que está optimizado para la tecnología moderna de redes. Llevando los beneficios de la tecnología moderna de redes al mundo de los instrumentos musicales electrónicos, las ventajas de OSC incluyen interoperabilidad, precisión, flexibilidad y una mejor organización y documentación.

Este protocolo simple pero potente proporciona todo lo necesario para el control en tiempo real del sonido y el procesamiento de otros medios, sin dejar de ser flexible y fácil de implementar.

4.6.5.2 ¿Cómo funciona?

La unidad de transmisión de OSC es un paquete OSC. Cualquier aplicación que envía paquetes OSC es un cliente OSC; cualquier aplicación que recibe paquetes OSC es un servidor OSC.

Un paquete OSC consta de su contenido, un bloque contiguo de datos binarios y su tamaño, el número de bytes de 8 bits que comprende el contenido. El tamaño de un paquete OSC es siempre un múltiplo de 4.

La red subyacente que entrega un paquete OSC es responsable de entregar tanto el contenido como el tamaño a la aplicación OSC. Un paquete OSC puede representarse naturalmente

mediante un datagrama mediante un protocolo de red como UDP. En un protocolo basado en flujo, como TCP, el flujo debe comenzar con un int32 que indique el tamaño del primer paquete, seguido del contenido del primer paquete, seguido del tamaño del segundo paquete, etc.

El contenido de un paquete OSC debe ser un mensaje OSC o un paquete OSC. El primer byte del contenido del paquete distingue inequívocamente entre estas dos alternativas.

Un mensaje OSC consiste en un patrón de dirección OSC seguido de una cadena de etiqueta de tipo OSC seguida de cero o más argumentos OSC.

Un patrón de dirección OSC es una cadena OSC que comienza con el carácter '/' (barra diagonal).

Una cadena de etiqueta de tipo OSC es una cadena de OSC que comienza con el carácter ',' (coma) seguido de una secuencia de caracteres que corresponde exactamente a la secuencia de argumentos OSC en el mensaje dado.

Cada carácter después de la coma se denomina etiqueta de tipo OSC y representa el tipo del argumento OSC correspondiente.

4.6.5.3 MuseScore

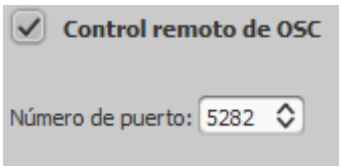
MuseScore es un programa de notación musical disponible para Linux, Mac OS X y Windows. Se trata de un editor con soporte completo para reproducir partituras e importar o exportar MusicXML y archivos MIDI estándar.

MuseScore es un software libre publicado bajo la licencia publica general de GNU, desarrollado en C++ y Qt.

Este software resulta de gran utilidad, ya que, posee la opción de establecer comunicación OSC entre el programa de notación y otro terminal externo, como un sintetizador o instrumento MIDI o, por otro lado, otra aplicación mismamente.

Para establecer comunicación es necesario habilitar en la sección de preferencias de su configuración el puerto por el que se entablará la transmisión. Por defecto, MuseScore utiliza para este tipo de procesos el puerto 5282.

Ilustración lii. Puerto por defecto de MuseScore para comunicación OSC



Una vez que se conocen los conceptos básicos para implementar el sistema requerido, se procede a la sinterización de todas las secciones, referentes a la aplicación web, comentadas previamente aplicándose al sistema final propuesto en este proyecto.

4.6.6 NAVEGADOR WEB

El navegador elegido para la monitorización e interacción con el cliente ha sido Mozilla Firefox.

Es un navegador web gratuito y de código abierto desarrollado por la Fundación Mozilla y su subsidiaria, Mozilla Corporation. Firefox está oficialmente disponible para Windows 7 o posterior, macOS y Linux.

Se trata de un navegador muy cómodo y completo en cuanto a desarrollo de aplicaciones se refiere.

Posee una cantidad de permisos y seguridad necesarias que lo diferencian notablemente de los navegadores convencionales más usados de Google o Microsoft, entre otros.

4.6.7 IMPLEMENTACIÓN

En primer lugar, se ha instalado y configurado el servidor Flask.

Los archivos que se van a necesitar para la demostración del sistema final serán:

- El archivo de audio de la grabación interpretada que será usado como elemento principal de navegación en el reproductor web creado, así como la entrada del sistema de transcripción que obtendrá las secuencias pertinentes para su posterior sincronización.
- El archivo XML de la partitura original del que se obtendrá su archivo MIDI asociado, que será importado en MuseScore para posteriormente sincronizarlo con el archivo de audio importado en el reproductor web, mediante la comunicación OSC. También a partir de este se obtendrá un archivo XML, con extensión ".mpos" asociado al MIDI original, que indica la posición del compás que corresponde cada instante de tiempo de la partitura.

Para desarrollar la app web, se ha creado un reproductor web capaz de navegar en la reproducción, yendo de un instante de tiempo a otro si es preciso o pudiendo dejar correr la reproducción en su orden natural. Para ello, se ha creado un modelo web típico (MVC), montado sobre el framework Flask detallado previamente.

Flask será el encargado de organizar y controlar el proceso web.

La estructura de la Flask app en cuanto a directorios será:

- **APP FLASK**
  - App.py
  - Transcript.py
  - Align.py
  - Statics
    - Archivos JavaScript
    - Archivos CSS
    - Otros archivos necesarios
  - Templates
    - Archivos HTML
  - Uploads
    - Archivos multimedia importados por el cliente

App.py: se trata del controlador de la aplicación, el archivo principal desarrollado en Python con el que se llamarán a todas las funciones necesarias.

Transcript.py / Align.py: se tratan de los módulos de transcripción y alineamiento musical con los que mediante las llamadas del app.py se calcularán las predicciones y por consiguiente las secuencias de alineamiento necesarias para el sincronismo.

Statics: directorio en el cual se encontrarán todos los archivos del servidor con los que se dará funcionalidad web, estética y dinamismo a la aplicación.

Templates: en este folder aparecerán todos los archivos HTML necesarios para la representación de la app que serán renderizados desde el controlador.

Uploads: en esta carpeta, a priori vacía, serán almacenados los archivos multimedia que el cliente importará y que, por tanto, el servidor se encargará de su correcto uso.

Una vez explicada la arquitectura de la aplicación será necesario detallar el proceso que realizará esta para llegar a sincronizar finalmente los resultados obtenidos con la partitura original.

4.6.8 Funcionamiento. Proceso secuencial.

En primer lugar, se ejecuta el archivo app.py en el equipo servidor que activará la aplicación y será accesible desde la dirección htpps:127.0.0.1:5000.

Una vez que el cliente acceda a dicha dirección local se abrirá la aplicación con un archivo HTML renderizado en el que se podrá importar los archivos necesarios (audio de Interpretación de la pieza + XML de la partitura original).

Antes de seguir avanzando con el funcionamiento del interfaz web será necesario que el cliente active los puertos, ejecutando en segundo plano el módulo de OSC mediante NodeJs, para hacer posible la comunicación OSC con MuseScore, el cual también será iniciado.

A continuación, el cliente importará los datos solicitados y los enviará a la carpeta de almacenamiento “uploads”. Por consiguiente, el servidor tomará el archivo de audio importado y lo enviará al módulo de transcripción desde el controlador. Transcurrido un breve espacio de tiempo, se finalizará la predicción de la interpretación y será llamado el módulo de alineamiento, siendo los parámetros de entrada: la matriz previamente estimada por el predictor y el MIDI generado por el XML importado por el cliente.

Una vez finalizado el procesamiento se obtienen dos secuencias de alineamiento, “p” y “q”, las cuales serán los datos con los que se realizará la sincronización.

Por tanto, el servidor renderizará el HTML correspondiente al reproductor web desarrollado, introduciendo también las secuencias anteriormente exportadas.

El formato del reproductor web será el típicamente usado para cualquier página web dinámica, es decir, estará formado por el propio HTML, un archivo de estilo CSS así como los correspondientes archivos JavaScript que realizarán todas y cada una de las funcionalidades del reproductor, eso sí, todos estos documentos extraídos del folder “statics”.

El estilo de la interfaz web a nivel de cliente modelado mediante el archivo CSS, dará forma a aspectos estéticos como: el tipo de letra, colores, así como la estructura del reproductor, la posición de los elementos en la página, etc.

Por otro lado, además se importa un archivo JavaScript, desarrollado para darle funcionalidad al reproductor, esta será la parte más compleja del interfaz web, por lo que se introducirá más en detalle.

El archivo JavaScript (audio.min.js) desarrollado para crear los módulos, que requiere el reproductor y el sistema, posee varias funciones indispensables para el correcto uso del mismo. Mediante este script va ser capaz de reproducir cualquier archivo importado desde el equipo de manera que se irá obteniendo el instante exacto de reproducción, en tiempo real, para la obtención posterior del compás al que va perteneciendo en la partitura original.

También, será posible el desplazamiento de la reproducción (“scroll”), es decir, se podrá navegar a través del reproductor, yendo a cualquier instante de tiempo, volviendo al principio, al final, etc, de manera que se irán obteniendo estos valores instantáneos de tiempo mediante una función asíncrona.

Una vez que se obtiene el valor de tiempo por el cual transcurre la reproducción de la interpretación, se interpola este valor mediante las secuencias de alineamiento, calculadas previamente desde el servidor, obteniendo el valor de tiempo al que corresponde en el archivo original. Teniendo esto, a través del archivo XML, con extensión “.mpos” asociado al MIDI de la partitura, se ha realizado un sentencia para leer este archivo y obtener en cada intervalo de tiempo el compás al que corresponde en la partitura.

Mediante la comunicación OSC, se va transmitiendo simultáneamente el valor del compás en cada instante a través del puerto por defecto de MuseScore, es decir, el puerto 5282. El formato del mensaje será de la forma “’/select-measure’, compas”, y será enviado cada vez que vaya cambiando el segundero del reproductor.

Para que se ejecute la transmisión es necesario crear un servidor Websocket con NodeJs, que se encargará de abrir el puerto predeterminado y establecer la conexión UDP entre el navegador web y MuseScore.

Una vez que existe comunicación entre la aplicación web con el programa de notación musical, se irá mostrando en la partitura el compás por el que transcurre la interpretación predicha, mediante el indicador de compás de MuseScore.

Con esto se finalizará el proceso de demostración del sistema final propuesto.

Todo el código relacionado con el desarrollo de la aplicación y del sistema se encuentra clonado en un repositorio en Github. Se puede descargar accediendo al siguiente enlace:

https://github.com/Franciscocartas5/TFG_repo.git

5. CONCLUSIONES

A modo de conclusión de este proyecto, es preciso apuntar que se han alcanzado cada uno de los propósitos de desarrollo en los que se basa este sistema creado. Se han conseguido los objetivos con los que se comenzó a realizar este sistema, profundizando en la tecnología de aprendizaje automático con redes neuronales desarrollado principalmente en Python.

Los resultados obtenidos del sistema han sido bastante notables, resaltando además el modo de trabajo en remoto usado, debido a la necesidad de potencia computacional que requiere este tipo de tecnología que trabaja con una cantidad inmensa de datos e iteraciones, con el que se han ido guardando los diversos modelos de entrenamiento obtenidos y prediciendo la señal de estimación necesaria para la revisión de los resultados.

Para finalizar, se ha de decir que este inmenso y, aún, desconocido campo de la tecnología de aprendizaje automático e inteligencia artificial, está comenzando a entrar en aspectos cotidianos y va a ir en aumento a medida que crezcan las tecnologías que lo soportan, equipos, almacenamiento de datos, entre otros. Esta inclusión tecnológica puede ser de gran utilidad en cualquier tipo de situación natural del día a día usada correctamente, como, por ejemplo, en hospitales, en carreteras, en los hogares, en educación, en las industrias, etc.

Por lo tanto, será de gran importancia y repercusión investigar a fondo cualquier aspecto afín para mejorar los resultados en la medida de lo posible y así tener en cuenta las decisiones que puede ofrecer este tipo de tecnologías para llegar a la respuesta final con un punto de objetividad basado en el análisis probabilístico de millares de datos, dándole por supuesto también el enfoque racional a la solución final del problema en cuestión, interpretando cada uno de los resultados obtenidos.

6. REFERENCIAS BIBLIOGRÁFICAS

Referencias de internet:

https://docs.python.org

https://wiki.python.org

https://www.bigdataframework.org

https://www.analyticsvidhya.com

https://keras.io

http://www.piano-midi.de

http://developer.mozilla.org

https://www.coursera.org/learn/neural-networks-deep-learning

https://flask.palletsprojects.com/en/1.1.x/

Referencias de libros:

Multipitch estimation of piano sounds using a new probabilistic spectral smoothness principle, V. Emiya, R. Badeau, B. David, IEEE Transactions on Audio, Speech and Language Processing.

Colin Raffel and Daniel P. W. Ellis. Intuitive Analysis, Creation and Manipulation of MIDI Data with pretty_midi. In 15th International Conference on Music Information Retrieval Late Breaking and Demo Papers, 2014.