UNIVERSIDAD AUÓNOMA DE GUADALAJARA

CON RECONOCIMIENTO DE VALIDEZ OFICIAL DE ESTUDIOS DE LA SECRETARÍA DE EDUCACIÓN PÚBLICA SEGÚN ACUERDO No.158 DE FECHA 17 DE JULIO DE 1991

POSTGRADO E INVESTIGACIÓN

Composición Musical utilizando redes de Deep Learning Long Short Term Memory

TRABAJO DE INVESTIGACIÓN

QUE PRESENTA

EFRAIN ADRIAN LUNA NEVAREZ

PARA OBTENER EL GRADO DE

MAESTRÍA EN CIENCIAS COMPUTACIONALES

DIRECTOR Y ASESOR:

MTRO. JUAN ANTONIO VEGA FERNANDEZ

GUADALAJARA, JALISCO. AGOSTO 2018

TTS-03-185-0731-18-045

Dedicatoria

Este trabajo se lo dedico a todos los músicos que sean apasionados por la tecnología, y ven en ella una forma para seguir evolucionando musicalmente.

Agradecimientos

Primeramente agradezco a Dios por permitirme concluir un ciclo más en mi formación profesional, cada paso que doy siempre es gracias a él.

Agradezco a mi madre por enseñarme el valor del trabajo y del estudio. Gracias a ella que me brindo un apoyo cuando flaqueaba y supo guiarme en este proceso.

A mi novia que estuvo conmigo en épocas difíciles de mi vida, cuando pensé que no podría terminar con mis estudios por todas las responsabilidades que había adquirido, ella siempre estuvo ahí para animarme a seguir adelante.

Mis hermanas las cuales siempre me dieron palabras de aliento y estuvieron preocupadas por mi, en todo momento, agradezco su apoyo incondicional.

Al CONACYT porque me brindo el apoyo económico para poder terminar este grado de estudio.

A mis maestros que tuvieron la suficiente paciencia y dedicación para transmitirnos parte de su conocimiento, gracias a ellos pude concluir con este grado, ganando el suficiente conocimiento para poder aplicarlo en el campo laboral.

A mi Asesor el Maestro Juan Antonio Vega Fernández, el cual me guío a lo largo de esta tesis para realizar un trabajo que pueda dejar algo de conocimiento a la comunidad, agradezco la paciencia que me tuvo y el tiempo que me dedico para la revisión de este proyecto.

A mis compañeros de generación los cuales me enseñaron muchísimas cosas, fue muy enriquecedor contar con personas que a pesar de que habían vivido cosas muy diferentes y tenían en la mente seguir superándose escolarmente y profesionalmente hablando.

Finalmente agradezco a la UAG, escuela que me dejo tanto aprendizaje y vivencias. A todo el personal administrativo de esta institución que me guiaron a través de todos los trámites necesarios para concluir esta etapa.

Resumen

En este trabajo, se analizaron diferentes tipos de arquitecturas de redes de Deep Learning, con el fin de usarlas para crear nuevas piezas musicales.

Estas redes usan capas de neuronas Long Short Term Memory (LSTM), las cuales nos permitieron trabajar con secuencias de datos y guardar información a lo largo del tiempo.

Las redes descritas en este proyecto fueron entrenadas para reconocer y generar secuencias para los instrumentos de guitarra y bajo eléctrico.

El proceso de entrenamiento se logró usando una base de datos de archivos MIDI, los cuales son de genero rock y pop, por lo tanto, el aprendizaje de estas redes únicamente será para reconocer estos géneros musicales.

Las arquitecturas usadas fueron entrenadas para reconocer tonalidades de las canciones y en base a ellas, generar nuevas piezas musicales.

Tabla de Contenido

Lista de Figuras

Lista de Tablas

Capítulo 1
Introducción

La música es el idioma universal, no importa en qué lugar físico nos encontremos, todos alguna vez hemos escuchado una canción que se nos queda grabada en la cabeza por mucho tiempo. En ocasiones nos preguntamos cómo es que el autor compuso esa canción.

Existen muchos géneros musicales, cada uno cuenta con sus propias reglas, ritmos y sonidos característicos, por lo que los compositores que se dedican a crear nuevas canciones en un género en específico, deben de tener una gran noción acerca del mismo, así como sus generalidades. Es por eso que nos ponemos a pensar si una computadora es capaz aprender a reconocer las generalidades de un género musical y generar nuevas piezas musicales en base a esto.

Si bien es cierto, el componer una canción es un arte, pero las canciones también se pueden interpretar en forma matemática, esto nos abre la posibilidad de usar algoritmos computacionales para la creación de nuevas canciones.

1.1. Descripción del Problema

Los principales problemas en la composición musical son: la falta de ideas y el tiempo requerido para la composición.

Existen compositores que a lo largo del tiempo no les gusta explorar nuevas formas o ritmos musicales y eso ocasiona que sus composiciones sean muy similares entre sí.

El tiempo requerido para la creación de una nueva canción es muy variado, ya que depende del género, el número de instrumentos y la complejidad de la composición.

En este trabajo se busca usar un algoritmo computacional, que nos permita generar nuevas canciones de una manera más rápida y sencilla.

1.2. Objetivos

1.2.1. Objetivo General

Creación de un programa que use algoritmos de Deep Learning para servir de apoyo en una composición musical.

1.2.2. Objetivos Específicos

1.- Implementar una red neuronal de Deep Learning que sea capaz de crear melodías y armonías musicales en guitarra.

2.- Implementar una red neuronal de Deep Learning que sea capaz de crear melodías musicales en bajo eléctrico.

3.- Conjuntar la salida de las redes neuronales para generar un solo archivo de audio.

4.- Verificar la Tonalidad de la canción resultante.

5.- Comparar las diferentes arquitecturas.

1.3. Justificación

El uso de algoritmos de Deep Learning en la música no es algo nuevo, sin embargo, existen muchas áreas de oportunidad en cuestión de la implementación de estos algoritmos.

En el articulo "Deep Learning Techniques for Music Generation - A Survey" [6] podemos observar diferentes maneras para crear música con redes neuronales de Deep Learning. Este es uno de los trabajos más completos que existen hasta la fecha. Los autores de este articulo comparan diferentes estilos de redes (CNN, RNN y generativas), para ver cual es la que mejor para creación musical. Aunque es un articulo muy completo no se aborda como hacer que las redes aprendan a distinguir tonalidades.

El articulo "Deep Learning for Music" [4] presenta una buena forma para la creación de armonías y melodías musicales utilizando redes de Deep Learning y archivos en formato MIDI. En este trabajo hacen falta maneras para validar la salida musical.

La mayor parte de los trabajos en esta área están basados en melodías y armonías de piano, sin embargo es interesante experimentar con otros instrumentos, ya que cada uno cuenta con su propia manera de interpretar una pieza musical.

Lo que se pretende en esta investigación es tener una manera fácil y rápida para la creación de nuevas composiciones musicales de guitarra y bajo.

1.4. Delimitación

Este proyecto se centrara en crear melodías y armonías musicales de guitarra y bajo eléctrico por lo que no se analizaran otros instrumentos.

Las nuevas canciones creadas por estos algoritmos no buscan ser ideas finales de composición, la única intención es crear una serie de ideas para nuevas canciones.

La base de datos usada para entrenar a la red neuronal consta de canciones de género rock y pop únicamente, no se entrenara la red para reconocer y generar canciones de otros géneros musicales.

1.5. Organización de la Tesis

Esta tesis se divide en varios capítulos, los cuales se describen a continuación:

- **Capítulo 1.-** Se presenta una descripción del problema, justificación, delimitación y antecedentes históricos de la investigación.
- **Capítulo 2.-** Contiene el marco teórico de las redes neuronales, su clasificación y la manera de entrenarlas.
- **Capítulo 3.-** Aborda el marco teórico musical, todos los conceptos básicos para interpretar y leer música.
- **Capítulo 4.-** Se muestra el desarrollo del proyecto, la arquitectura y los componentes del sistema.
- **Capítulo 5.-** Contiene los resultados de las diferentes etapas del proyecto.
- **Capítulo 6.-** Se muestran las conclusiones del proyecto, así como también los trabajos futuros que podrían realizarse.

1.6. Antecedentes históricos

A lo largo del tiempo han surgido varios estudios referentes al uso de algoritmos de Deep Learning con la música, a continuación se presentan algunos de los artículos más relevantes en este tema:

1.6.1. Unsupervised Feature Learning for Audio Classification Using Convolutional Deep Belief Networks [1]

"En los últimos años, los enfoques de Deep Learning han ganado un gran interés como una forma de construir representaciones jerárquicas a partir de datos sin etiquetar. Sin embargo, según nuestro conocimiento, estos enfoques de Deep Learning no se han estudiado exhaustivamente para datos auditivos. En este documento, aplicamos redes de creencias profundas convolucionales a los datos de audio y los evaluamos empíricamente en varias tareas de clasificación de audio. Para el caso de datos de voz, mostramos que las características aprendidas corresponden a teléfonos / fonemas. Además, nuestras representaciones de funciones entrenadas a partir de datos de audio sin etiquetar muestran un rendimiento muy bueno para múltiples tareas de clasificación de audio. Esperamos que este documento inspire más investigación sobre los enfoques de aprendizaje profundo aplicados a una amplia gama de tareas de reconocimiento de audio."

1.6.2. Feature Learning and Deep Architectures: New Directions for Music Informatics [2]

"A medida que buscamos avanzar en el estado del arte en informática de música basada en contenido, hay un sentido general de que el progreso se está desacelerando en todo el campo. En una inspección más cercana, las trayectorias de rendimiento en varias aplicaciones revelan que este es realmente el caso, lo que plantea algunas preguntas difíciles para la disciplina: ¿por qué nos estamos desacelerando y qué podemos hacer al respecto? Aquí, nos esforzamos por abordar estas dos preocupaciones. Primero, revisamos críticamente el enfoque estándar para el análisis de la señal de música e identificamos tres deficiencias específicas de los métodos actuales: el diseño de características artesanal es subóptimo e insostenible, el poder de las arquitecturas

poco profundas es fundamentalmente limitado y el análisis a corto plazo no puede codificar musicalmente estructura significativa Reconociendo los avances en otros dominios perceptivos de inteligencia artificial, ofrecemos que el aprendizaje profundo tiene el potencial de superar cada uno de estos obstáculos. A través de argumentos conceptuales para el aprendizaje de características y arquitecturas de procesamiento más profundas, demostramos cómo los modelos de procesamiento profundo son extensiones más poderosas de los métodos actuales y por qué ahora es el momento de este cambio de paradigma. Finalmente, concluimos con una discusión de los desafíos actuales y el impacto potencial para motivar aún más una exploración de esta área de investigación prometedora."

1.6.3. Improving Content-based and Hybrid Music Recommendation Using Deep Learning [3]

"Los sistemas de recomendación de música basados en contenido existentes suelen emplear un enfoque textit two-stage. Primero extraen características de contenido de audio tradicionales como los coeficientes cepstral de frecuencia de Mel y luego predicen las preferencias del usuario. Sin embargo, estas características tradicionales, originalmente no creadas para la recomendación de música, no pueden capturar toda la información relevante en el audio y, por lo tanto, limitan el rendimiento de la recomendación. Usando un modelo novedoso basado en una red de creencias profundas y un modelo gráfico probabilístico, unificamos las dos etapas en un proceso automatizado que aprende simultáneamente las características del contenido de audio y hace recomendaciones personalizadas. Comparado con los modelos basados en Deep Learning existentes, nuestro modelo supera a los de las etapas de arranque en caliente y de arranque en frío sin depender del filtrado colaborativo (CF). Luego presentamos un método híbrido eficiente para integrar a la perfección las características aprendidas automáticamente y la CF. Nuestro método híbrido no solo mejora significativamente el rendimiento de la FQ, sino que también supera el método híbrido basado en características tradicionales."

1.6.4. Deep Learning for Music [4]

"Nuestro objetivo es poder construir un modelo generativo a partir de una arquitectura de red neuronal profunda para intentar crear música que tenga armonía y melodía y que sea pasable como música compuesta por humanos. El trabajo anterior en la generación de música se ha centrado principalmente en crear una sola melodía. El trabajo más reciente sobre el modelado de música polifónica, centrado alrededor de la estimación de densidad de probabilidad de series de tiempo, ha logrado cierto éxito parcial. En particular, ha habido mucho trabajo basado en redes neuronales recurrentes combinadas con máquinas de Boltzmann restringidas (RNN-RBM) y otros modelos similares basados en energía recurrente. Sin embargo, nuestro enfoque es realizar el aprendizaje y la generación de extremo a extremo solo con redes neuronales profundas."

1.6.5. End-to-end learning for music audio tagging at scale [5]

"La falta de datos tiende a limitar los resultados de la investigación de Deep Learning, especialmente cuando se trata de acumulaciones de aprendizaje de extremo a extremo que procesan datos sin procesar, como las formas de onda. En este estudio, 1.2M pistas comentadas con etiquetas musicales están disponibles para entrenar a nuestros modelos de extremo a extremo. Esta gran cantidad de datos nos permite explorar sin restricciones dos paradigmas de diseño diferentes para el etiquetado automático de música: modelos sin supuestos: el uso de formas de onda como entrada con filtros convolucionales muy pequeños; y modelos que se basan en el conocimiento del dominio: espectrogramas log-mel con una red neuronal convolucional diseñada para aprender las características tímbricas y temporales. Nuestro trabajo se centra en estudiar cómo funcionan estos dos tipos de arquitecturas profundas cuando se dispone de conjuntos de datos de tamaño variable para la capacitación: el MagnaTagATune (canciones de 25 k), el conjunto de datos de la canción Million (canciones de 240 k) y un conjunto de datos privado de 1,2 millones de canciones. Nuestros experimentos sugieren que las suposiciones del dominio de la música son relevantes cuando no hay suficientes datos de entrenamiento disponibles, lo que demuestra que los modelos basados en formas de onda superan a los basados en espectrogramas en escenarios de datos a gran escala."

1.6.6. Deep Learning Techniques for Music Generation - A Survey [6]

"Este documento es una encuesta y un análisis de diferentes formas de utilizar el Deep Learning (redes neuronales artificiales profundas) para generar contenido musical."

"Proponemos una metodología basada en cinco dimensiones para nuestro análisis: - Objetivo - ¿Qué contenido musical se generará? Por ejemplo, melodía, polifonía, acompañamiento y contrapunto: ¿para qué destino y para qué uso? Para ser realizado por un humano (s) o por una máquina. - Representación - ¿Cuáles son los conceptos a manipular? Por ejemplo, la forma de onda, el espectrograma, la nota, el acorde, el medidor y el tiempo: ¿qué formato se utilizará? Por ejemplo, MIDI, piano roll y texto. ¿Cómo se codificará la representación? Por ejemplo, escalar, one-hot y many-hot. - Arquitectura - ¿Qué tipo de red neuronal profunda se va a utilizar? Por ejemplo, red feedforward, red recurrente, autocodificador y redes adversas generativas. - Retos - ¿Cuáles son las limitaciones índice y desafíos abiertos? Por ejemplo, variabilidad, interactividad y creatividad. - Estrategia - ¿Cómo modelamos y controlamos el proceso de generación? Por ejemplo, avance en un solo paso, avance en el decodificador, manipulación de muestreo y entrada. Para cada dimensión, realizamos un análisis comparativo de varios modelos y técnicas y proponemos una tipología multidimensional tentativa. Esta tipología es de abajo hacia arriba, basada en el análisis de muchos sistemas existentes basados en deep learning para la generación de música seleccionados de la literatura relevante. Estos sistemas se describen en esta encuesta/análisis y se utilizan para ejemplificar las diversas opciones de objetivos, representación, arquitectura, desafíos y estrategias. La parte final del documento incluye algunas discusiones y algunas perspectivas."

"Este artículo es una versión simplificada (DRM débil) del siguiente libro: Jean-Pierre Briot, Gaetan Hadjeres y Francois Pachet, Técnicas de aprendizaje profundo para la generación de música, Síntesis computacional y Sistemas creativos, Springer Nature, 2019."

Capítulo 2

Redes neuronales

Las redes neuronales artificiales son un conjunto de neuronas creadas artificialmente para el desarrollo de inteligencia artificial en una computadora.

Estas redes están basadas en las redes biológicas del cerebro humano, modelando todos los factores biológicos de las neuronas.

Debido a su diseño las redes son capaces de aprender de la experiencia, de generalizar de casos anteriores a nuevos casos, de abstraer características esenciales a partir de entradas que representan en ocasiones información irreverente.

La capacidad de aprendizaje adaptativo es una característica fundamental de las redes neuronales y les permiten llevar a cabo ciertas tareas mediante un entrenamiento previo, pueden aprender a diferenciar patrones y generalizar a partir de estos. Son considerados sistemas dinámicos ya que son capaces de adaptarse a nuevas condiciones de entrada.

Tienen una alta tolerancia a fallos ya que son capaces de detectar patrones aun cuando estos patrones posean ruido, distorsión o simplemente están incompletos. Estos programas son capaces de seguir funcionando incluso si parte de la red presente fallas.

La información se almacena de forma distribuida en las conexiones de las neuronas, provocando redundancia de información, es decir se guardara sus valores en base a la función de activación que posee cada neurona, de esta manera si una neurona es destruida o presenta fallas, las otras neuronas podrán aprender la información de la neurona que fallo.

Las neuronas humanas poseen diferentes secciones:

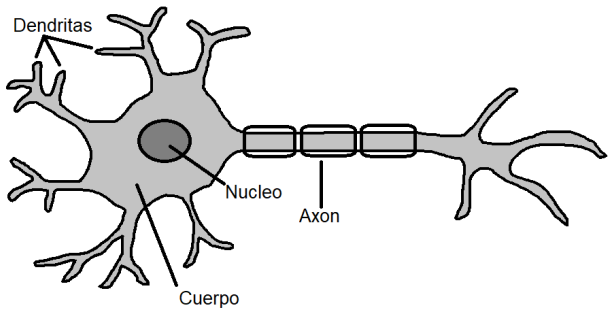

Figura 2.1: Estructura general de una neurona

En una neurona artificial se busca la emulación de las principales secciones de una neurona las cuales son:

- **Cuerpo.-** Se encarga de producir un impulso eléctrico en base a las entradas de la neurona.
- **Dendritas.-** son filamentos capaces de crear conexiones con otras neuronas.
- **Axón.-** Es el encargado de transmitir el impulso eléctrico generado por el cuerpo.

A continuación se muestra una imagen de como luciría una neurona artificial:

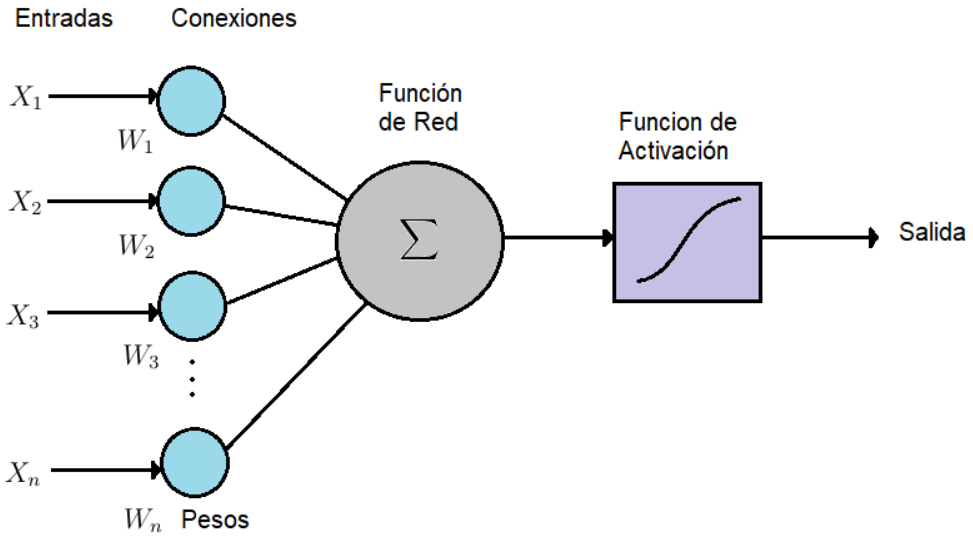

Figura 2.2: Neurona Artificial

Esta neurona posee las siguientes secciones:

- X_1, X_2, ..., X_n **.-** Son las entradas de la neurona.
- W_1, W_2, ..., W_n**.-** Pesos específicos que tendrá cada entrada, esto hace que las entradas no valgan lo mismo ponderadamente.
- **Función de red.-** Esta es una función de sumatoria de las entradas.
- **Función de activación.-** Si la suma de las entradas es mayor o igual que el umbral definido por esta función se tendrá una señal a la salida.

La salida de la neurona viene dada por esta ecuación:

$$y_j = f(\sum_{i=1}^{n}(W_{ij}x_i + \theta_j)) \tag{2.0.1}$$

Una red neuronal no es más que la interconexión de varias neuronas artificiales, en la cual podemos identificar al menos tres secciones:

- **Capa de entrada.-** En esta capa se procesan todas las entradas, y si estas entradas son capaces de excitar las neuronas de esta capa se producirá una señal de salida.
- **Capa oculta.-** En esta capa se encuentran las neuronas encargadas del aprendizaje de la red.
- **Capa de salida.-** Esta neurona o neuronas de salida tendrán la salida del sistema.

La forma en que las redes aprenden es mediante la modificación de los pesos de las entradas.

Las redes neuronales artificiales han ido evolucionando con el paso del tiempo, hoy en día existen muchos modelos de redes neuronales, todas ellas con ventajas y desventajas si son comparadas entre ellas.

2.1. Aprendizaje de las redes neuronales

El procedimiento utilizado para llevar a cabo el proceso de aprendizaje en una red neuronal se denomina entrenamiento.

El problema de aprendizaje en las redes neuronales se formula en términos de la minimización de la función de error (o pérdida) asociada.

Normalmente, esta función está compuesta por dos términos, uno que evalúa cómo se ajusta la salida de la red neuronal al conjunto de datos de que disponemos, y que se denomina término de error, y otro que se denomina término de regularización, y que se utiliza para evitar el sobreaprendizaje por medio del control de la complejidad efectiva de la red neuronal.

Por supuesto, el valor de la función de error depende por completo de los parámetros de la red neuronal: los pesos sinápticos entre neuronas, y los bias asociados a ellas, que, como suele ser ya habitual, se pueden agrupar adecuadamente en un único vector de pesos de la dimensión adecuada, que denotaremos por w. En este sentido, podemos escribir $f(w)$ para indicar que el valor del error que comete la red neuronal depende de los pesos asociados a la misma. Con esta formalización, nuestro objetivo es encontrar el valor w^* para el que se obtiene un mínimo global de la función f, convirtiendo el problema de aprendizaje en un problema de optimización.

En general, la función de error es una función no lineal, por lo que no disponemos de algoritmos sencillos y exactos para encontrar sus mínimos. En consecuencia, tendremos que hacer uso de una búsqueda a través del espacio de parámetros que, idealmente, se aproxime de forma iterada a un (error) mínimo de la red para los parámetros adecuados.

De esta forma, se comienza con una red neuronal con algún vector inicial de parámetros (a menudo elegido al azar), a continuación se genera un nuevo vector de parámetros, esperando que con ellos la función de error se reduzca (aunque dependiendo del método elegido, no es obligatorio, y temporalmente se puede admitir un empeoramiento del error siempre y cuando conduzca a una disminución posterior más acusada). Este proceso se repite, normalmente, hasta haber reducido el error bajo un umbral tolerable, o cuando se satisfaga una condición específica de parada.

El Descenso del Gradiente es el algoritmo de entrenamiento más simple y también el más extendido y conocido. Solo hace uso del vector gradiente, y por ello se dice que es un método de primer orden.

Este método para construir el punto w_{i+1} a partir de w_i se traslada este punto en la dirección de entrenamiento $d_i = -g_i$. Es decir:

$$w_{i+1} = w_i - g_i v_i \tag{2.1.1}$$

Donde el parámetro v se denomina tasa de entrenamiento, que puede fijarse a priori o calcularse mediante un proceso de optimización unidimensional a lo largo de la dirección de entrenamiento para cada uno de los pasos (aunque esta última opción es preferible, a menudo se usa un valor fijo, $v_i = v$ con el fin de simplificar el proceso).

Aunque es muy sencillo, este algoritmo tiene el gran inconveniente de que, para funciones de error con estructuras con valles largos y estrechos, requiere muchas iteraciones. Se debe a que, aunque la dirección elegida es en la que la función de error disminuye más rápidamente, esto no significa que necesariamente produzca la convergencia más rápida.

Por ello, es el algoritmo recomendado cuando tenemos redes neuronales muy grandes, con muchos miles de parámetros, ya que sólo almacena el vector gradiente (de tamaño n).

2.2. Deep Learning

Deep Learning usa redes neuronales con muchas capas para lograr aprendizajes más complejos. Este comportamiento asemeja la forma en que el cerebro humano toma decisiones, el cual usa la interconexión de varias capas de neuronas para realizar actividades complejas. Dentro de las redes de Deep Learning se tienen 2 tipos muy usados en la actualidad:

- **Redes convolucionales (CNN).-** Este tipo de redes usa la convolución en varias de sus capaz para lograr el procesamiento de parámetros que se pueden representar en un espacio R^2, un ejemplo claro de esto son las imágenes y vídeos, por lo tanto si se quiere hacer una clasificación o reconocimiento de imágenes, este tipo de redes nos proporcionan una buena herramienta de procesamiento.

- **Redes recurrentes (RNN).-** Este tipo de redes son muy usadas cuando se busca analizar una secuencia de datos, estas redes poseen memoria y una retroalimentación de la salida a la entrada.

2.3. Redes neuronales recurrentes

La idea detrás de las RNN es hacer uso de la información secuencial. En una red neuronal tradicional suponemos que todas las entradas (y salidas) son independientes entre sí. Pero para muchas tareas eso es una muy mala idea. Si quieres predecir la siguiente palabra en una oración, es mejor que conozcas qué palabras vienen antes. Las RNN se llaman recurrentes porque realizan la misma tarea para cada elemento de una secuencia, y la salida depende de los cálculos previos. Otra forma de pensar acerca de las RNN es que tienen una "memoria"que captura información sobre lo que se ha calculado hasta ahora. En teoría, los RNN pueden hacer uso de la información en secuencias arbitrariamente largas, pero en la práctica se limitan a mirar hacia atrás solo unos pocos pasos.

La decisión de una red recurrente alcanzada en el paso de tiempo t-1 afecta la decisión que alcanzará un momento más tarde en el paso de tiempo t (Figura 2.3). Entonces, las redes recurrentes tienen dos fuentes de entrada, el presente y el pasado reciente, que se combinan para determinar cómo responden a los datos nuevos, de forma similar a como lo hacemos en la vida.

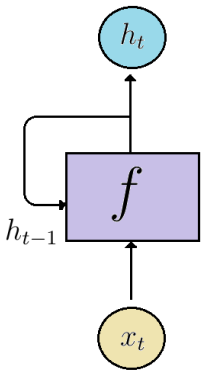

Figura 2.3: Redes recurrentes

Esa información secuencial se conserva en el estado oculto de la red recurrente, que logra abarcar muchos pasos de tiempo a medida que avanza para afectar el procesamiento de cada nuevo ejemplo. Está encontrando correlaciones entre eventos separados por muchos momentos, y estas correlaciones se llaman "dependencias a largo plazo", porque un evento en el tiempo depende de, y es una función de, uno o más eventos que vinieron antes. Una forma de pensar acerca de las RNN es esta: son

una forma de compartir pesos a lo largo del tiempo.

Así como la memoria humana circula invisiblemente dentro de un cuerpo, afectando nuestro comportamiento sin revelar su forma completa, la información circula en los estados ocultos de las redes recurrentes.

Describiremos el proceso de llevar la memoria hacia adelante matemáticamente:

$$h_t = \phi(Wx_t + Uh_{t-1}) \tag{2.3.1}$$

El estado oculto en el paso de tiempo t es h_t. Es una función de la entrada al mismo tiempo paso x_t, modificada por una matriz de ponderación W (como la que usamos para las redes feedforward) agregada al estado oculto del paso de tiempo anterior h_{t-1} multiplicado por su propio estado oculto matriz U de estado oculto, también conocida como matriz de transición y similar a una cadena de Markov. Las matrices de peso son filtros que determinan la importancia de acuerdo tanto con la entrada actual como con el estado oculto pasado. El error que generan volverá a través de la propagación inversa y se usará para ajustar sus ponderaciones hasta que el error no pueda bajar más.

Debido a que este ciclo de retroalimentación ocurre en cada paso de la serie, cada estado oculto contiene rastros no solo del estado oculto anterior, sino también de todos los que precedieron a h_{t-1} mientras la memoria pueda persistir.

2.3.1. LSTM

A mediados de los años 90, los investigadores alemanes Sepp Hochreiter y Juergen Schmidhuber propusieron una variación de la red recurrente con las denominadas unidades de memoria a largo plazo, o LSTM, como una solución al problema del gradiente de fuga.

Los LSTM ayudan a preservar el error que se puede volver a propagar a través del tiempo y las capas. Al mantener un error más constante, permiten que las redes recurrentes continúen aprendiendo durante muchos pasos de tiempo (más de 1000), abriendo así un canal para vincular causas y efectos de forma remota. Este es uno de los desafíos centrales para el aprendizaje automático y la IA, ya que los algoritmos se enfrentan con frecuencia a entornos en los que las señales de recompensa son dispersas y diferidas, como la vida misma.

Los LSTM contienen información fuera del flujo normal de la red recurrente en una

celda cerrada. La información puede almacenarse, escribirse o leerse desde una celda, al igual que los datos en la memoria de una computadora. La célula toma decisiones sobre qué almacenar y cuándo permitir las lecturas, escrituras y borraduras, a través de puertas que se abren y cierran. Sin embargo, a diferencia del almacenamiento digital en computadoras, estas puertas son análogas, implementadas con la multiplicación de elementos por sigmoides, que están todas en el rango de 0-1. Siendo analógica tiene la ventaja sobre digital de ser diferenciable y, por lo tanto, adecuado para la propagación inversa.

Esas puertas actúan sobre las señales que reciben, y de forma similar a los nodos de la red neuronal, bloquean o transmiten información en función de su fuerza e importación, que filtran con sus propios conjuntos de ponderaciones. Esos pesos, como los pesos que modulan los estados de entrada y ocultos, se ajustan a través del proceso de aprendizaje de redes recurrentes. Es decir, las células aprenden cuándo permiten que los datos entren, salgan o se eliminen a través del proceso iterativo de hacer conjeturas, volver a propagar el error y ajustar los pesos mediante el descenso del gradiente.

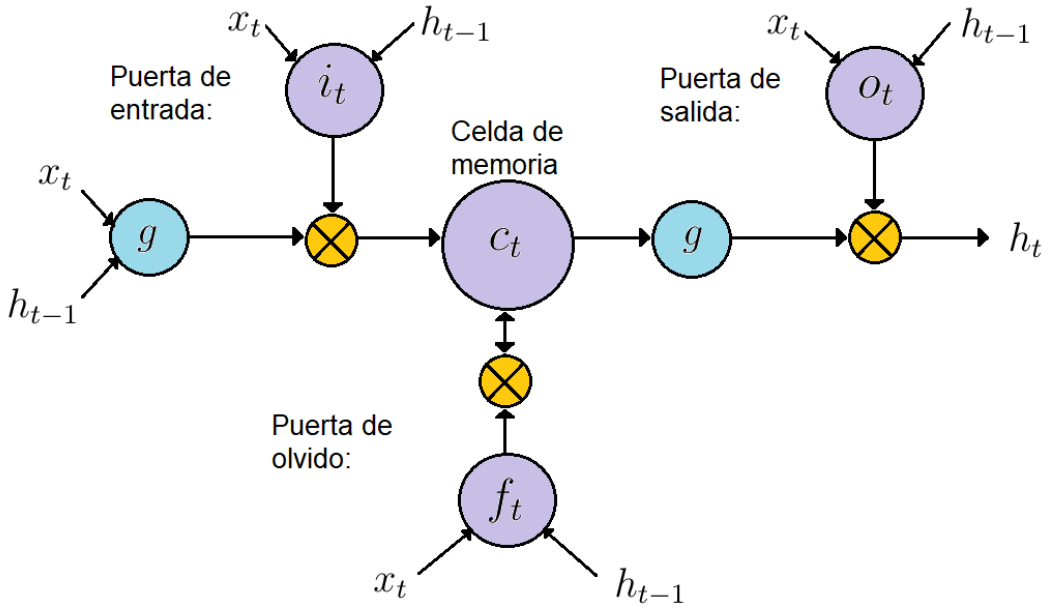

Figura 2.4: Redes LSTM

2.4. Optimizadores

Los algoritmos de optimización nos ayudan a minimizar (o maximizar) una función de objetivo $E(x)$, que es simplemente una función matemática que depende de los parámetros internos de aprendizaje del modelo que se utilizan para calcular los valores objetivo (Y) a partir de Conjunto de predictores (X) utilizados en el modelo.

Los algoritmos de optimización están en 2 categorías principales:

- **Algoritmos de optimización de primer orden:** estos algoritmos minimizan o maximizan una función de pérdida $E(x)$ utilizando sus valores de gradiente con respecto a los parámetros. El algoritmo de optimización de primer orden más utilizado es Descenso por Gradiente. La derivada de primer orden nos dice si la función está disminuyendo o aumentando en un punto en particular.
- **Algoritmos de optimización de segundo orden:** estos algoritmos utilizan la derivada de segundo orden, que también se llama Hessiana para minimizar o maximizar la función de error $E(x)$. La Hessiana es una matriz de derivados parciales de segundo orden. Dado que la segunda derivada es costosa de calcular, el segundo orden no se usa mucho. La derivada de segundo orden nos dice si la primera derivada está aumentando o disminuyendo, lo que sugiere la curvatura de la función.

Un gradiente es simplemente un vector que es una generalización multivariable de una derivada ($\frac{dy}{dx}$) en funciones multivariables.

2.4.1. Descenso por gradiente

El descenso por gradiente es la técnica más usada para entrenar y optimizar sistemas inteligentes. Es utilizada para realizar la actualización de los pesos en modelos de redes neuronales y minimizar la función de error $E(x)$, la forma en que actualiza los parámetros es utilizando la siguiente ecuación:

$$\theta = \theta - \eta \nabla J(\theta) \tag{2.4.1}$$

Donde η es la tasa de aprendizaje, y $\nabla J(\theta)$ es el gradiente de la función de error $J(\theta)$. El descenso por gradiente es mayormente usado para hacer la actualización de los pesos en una red neuronal.

2.4.1.1. Descenso por gradiente estocástico (SGD)

Este método realiza una actualización de parámetros para cada ejemplo de entrenamiento. Generalmente es una técnica mucho más rápida. Realiza una actualización a la vez.

Debido a estas actualizaciones frecuentes, las actualizaciones de parámetros tienen una gran variación y hacen que la función de error fluctúe a diferentes intensidades. Esto es realmente bueno porque nos ayuda a descubrir nuevos y posiblemente mejores mínimos locales, mientras que el Descenso de gradiente estándar solo convergerá al mínimo local.

El problema con la SGD es que, debido a las frecuentes actualizaciones y fluctuaciones, en última instancia, complica la convergencia al mínimo exacto y continuará superando debido a las frecuentes fluctuaciones.

$$\theta = \theta - \eta \nabla J(\theta; x(i); y(i)) \tag{2.4.2}$$

Donde $x(i), y(i)$ son ejemplos de entrenamiento.

2.4.1.2. Momento

Las oscilaciones de alta varianza en la SGD hacen que sea difícil alcanzar la convergencia, por lo que se inventó una técnica llamada Momentum que acelera la SGD al navegar a lo largo de la dirección relevante y suaviza las oscilaciones en direcciones irrelevantes. En otras palabras, todo lo que hace es agregar una fracción γ del vector de actualización del paso anterior al vector de actualización actual.

$$V(t) = \gamma V(t-1) + \eta \nabla J(\theta) \tag{2.4.3}$$

Para finalmente actualizar los parámetros por $\theta = \theta - V(t)$. El momento γ es usualmente puesto a 0.9 o un valor similar.

2.4.1.3. Gradiente acelerado Nesterov

El problema con el momento es que a medida que alcanzamos los mínimos, es decir, el punto más bajo de la curva, el impulso es bastante alto y no sabe disminuir la velocidad en ese punto debido al alto impulso podría hacer que se pierda por completo los mínimos y continuemos con la búsqueda.

Gradiente acelerado Nesterov (NAG) es una forma de dar a nuestro término de impulso este tipo de habilidad. Sabemos que usaremos nuestro término $\gamma V(t-1)$ para mover los parámetros θ. El cálculo de $\theta - \gamma V(t-1)$ nos da una aproximación de la siguiente posición de los parámetros. Ahora podemos mirar hacia adelante con eficacia calculando el gradiente pero teniendo en cuenta la posición futura.

$$V(t) = \gamma V(t-1) + \eta \nabla J(\theta - \gamma V(t-1)) \tag{2.4.4}$$

$$\theta = \theta - V(t) \tag{2.4.5}$$

2.4.1.4. Adagrad

Este método permite que la tasa de aprendizaje η se adapte en función de los parámetros. Así que hace grandes actualizaciones para parámetros poco frecuentes y pequeñas actualizaciones para parámetros frecuentes. Por esta razón, es adecuado para tratar con datos dispersos.

Adagrad modifica la tasa de aprendizaje general η en cada paso de tiempo t para cada parámetro θ en función de los gradientes pasados que se han calculado para θ.

$$\theta = \theta - \frac{\eta}{\sqrt{G + \epsilon}} g \tag{2.4.6}$$

2.4.1.5. AdaDelta

Es una extensión de AdaGrad que tiende a eliminar el problema de la tasa de aprendizaje en degradación. En lugar de acumular todos los gradientes cuadrados anteriores, Adadelta limita la ventana de gradientes pasados acumulados a un tamaño fijo w.

La suma de los gradientes se define recursivamente como una media decreciente de todos los gradientes cuadrados pasados. El promedio $E[g^2]_t$ en el tiempo t depende solo del promedio anterior y del gradiente actual.

$$E[g^2]_t = \gamma E[g^2]_{t-1} + (1-\gamma) g_t^2 \tag{2.4.7}$$

$$\Delta\theta_t = -\frac{RMS[\Delta\theta]_{t-1}}{RMS[g]_t} g_t \tag{2.4.8}$$

$$\theta_{t+1} = \theta_t + \Delta\theta_t \tag{2.4.9}$$

2.4.1.6. RMSprop

RMSprop y Adadelta se han desarrollado de forma independiente casi al mismo tiempo, debido a la necesidad de resolver las tasas de aprendizaje que disminuyen radicalmente en Adagrad.

RMSprop divide la velocidad de aprendizaje por un promedio decreciente exponencial de gradientes cuadrados.

$$E[g^2]_t = 0.9E[g^2]_{t-1} + 0.1g_t^2 \tag{2.4.10}$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t \tag{2.4.11}$$

2.4.1.7. Adam

La estimación del momento adaptativo (Adam) es otro método que calcula las tasas de aprendizaje adaptativo para cada parámetro. Además de almacenar un promedio de decaimiento exponencial de gradientes cuadrados pasados como AdaDelta, Adam también mantiene un promedio de decaimiento exponencial de gradientes pasados $\hat{m}_t$, similar al impulso.

$\hat{m}_t$ y $\hat{v}_t$ son valores del primer momento, que es la Media y el segundo momento, que es la varianza no centrada de los gradientes, respectivamente.

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \tag{2.4.12}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t} \tag{2.4.13}$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t \tag{2.4.14}$$

2.4.1.8. Nadam

Adam se puede ver como una combinación de RMSprop y momento: RMSprop contribuye con el promedio de decaimiento exponencial de los gradientes cuadrados pasados, mientras que el momento representa el promedio decreciente exponencial de los gradientes pasados.

Nadam (Estimación del momento adaptativo acelerado por Nesterov) por lo tanto, combina a Adán y NAG.

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon}(\beta_1 \hat{m}_t + \frac{(1-\beta_1)g_t}{1-\beta_1^t}) \tag{2.4.15}$$

Capítulo 3

Teoría musical

La teoría musical es el estudio de los elementos que conforman la música. En esta teoría se analizan todos los sonidos involucrados para la creación, análisis y composición musical.

La música es un arte que se basa en 2 elementos:

- Sonidos
- Silencios

Los sonidos tienen diferentes propiedades las cuales se describen a continuación:

- **Altura.-** un sonido puede ser agudo, medio o grave, dependiendo de la altura de su nota.
- **Duración.-** un sonido debe de tener una duración la cual se expresa en unidades de tiempo.
- **Intensidad.-** esto se refiere al volumen del sonido, puede ser débil o fuerte.
- **Timbre.-** se le conoce como timbre o color a un sonido con la misma nota que suena diferente dependiendo del instrumento usado para su interpretación.

Los silencios a su vez su única propiedad intrínseca es la duración, es decir en un silencio lo único que se mide es la duración del mismo.

3.1. Composición musical

La composición musical está catalogado como un arte que tiene como objeto crear nuevas piezas musicales.

Un músico puede optar por varios caminos para la creación de su obra, existen músicos que se basan en su simple sentido común y crean canciones líricamente, sin embargo el proceso formal de composición involucra todos los conceptos musicales básicos de la teoría musical, los cuales se describirán a continuación.

3.1.1. Notas musicales

Las notas es un sistema que se usa para la representación de los diferentes sonidos en la música. En el mundo occidental se usa un sistema de 12 notas, los cuales pueden ser repetidos con diferentes alturas para generar una gama muy amplia de sonidos. Dentro de este sistema de 12 notas tenemos las notas naturales y las notas con alteraciones. Las notas naturales son:

Do, Re, Mi, Fa, Sol, La, Si (7 notas)

En las notas con alteraciones se agregan o quitan medios tonos a una nota, para eso se usan los siguientes símbolos:

- **# (sostenido).-** agrega 1/2 tono a una nota.
- **x (doble sostenido).-** agrega 1 tono a una nota.
- ♭ **(bemol).-** disminuye 1/2 tono a una nota.
- ♭♭ **(doble bemol).-** disminuye 1 tono a una nota.

Usando los símbolos anteriores podemos definir las siguientes notas con alteraciones:

Do#, Re#, Fa#, Sol#, La# (5 notas)

Como se puede observar tanto Mi y Si no tienen sonidos con alteraciones ascendentes, ya que los sonidos producidos por estas alteraciones son igual al de las notas consecutivas, a este tipo de sonidos iguales se les conoce como notas enarmónicas.

Por ejemplo:

- Mi# sonaría exactamente igual que un Fa.
- Si# sonaría exactamente igual que un Do.

También las notas con alteraciones pueden ser representadas usando el símbolo de bemol (b), es decir restando medio tono a una nota. En este caso se tendrían las notas con alteraciones de la siguiente manera:

Re♭, Mi♭, Sol♭, La♭, Si♭ (5 notas)

Como se mencionó anteriormente, las notas que en sonido son exactamente iguales pero en nomenclatura son diferentes se conocen como notas enarmónicas, de tal manera que las 5 notas con alteraciones que se describieron se pueden hacer una comparación del sonido de las mismas, siendo así, tenemos que:

- Do# suena exactamente igual que Re♭.
- Re# suena exactamente igual que Mi♭.
- Fa# suena exactamente igual que Sol♭.
- Sol# suena exactamente igual que La♭.
- La# suena exactamente igual que Si♭.

Podemos ver la relación de tonos y semitonos (1/2 tonos) en la siguiente figura:

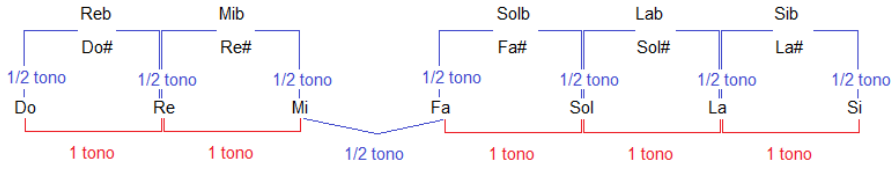

Figura 3.1: Relación de tonos y semitonos

Como se puede observar entre Mi y Fa existe un semitono, al igual que entre Si y Do.

El pentagrama es un sistema de cinco líneas y cuatro espacios para escribir música:

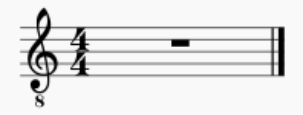

Figura 3.2: Pentagrama

Este sistema puede tener diferentes elementos dentro de los principales tenemos:

- Clave.
- Compás.
- Armadura.
- Notas.

3.1.2. Claves musicales

Las claves musicales se usan para darle nombre y altura a las notas musicales dentro de un pentagrama.

Las claves más usadas en la música son: la clave de Sol, la clave de Fa y la clave de Do.

La clave de Sol se usa en instrumentos agudos como la guitarra, violín, entre otros. Esta clave normalmente se escribe empezando en la segunda línea del pentagrama para formar una especie de G. El hecho de que esta clave se escriba en la segunda línea establece que esa línea será llamada como el nombre de la clave, en este caso Sol:

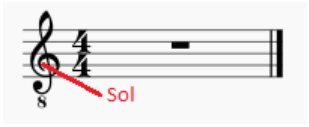

Figura 3.3: Clave de Sol en 2da. Linea

La clave de Fa es usada en instrumentos más graves, tales como el contrabajo, el bajo, etc. Normalmente se escribe empezando en la cuarta línea, de tal manera que esta línea tomara el nombre de Fa:

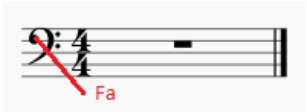

Figura 3.4: Clave de Fa en 4ta. Linea

La clave de Do es usada comúnmente para las voces, y esta clave se escribe en diferentes líneas dependiendo del timbre de la voz, para un tenor se usa la clave de Do en 4ta línea, mientras que para un soprano normalmente la clave se usa en 3ra o 2da línea:

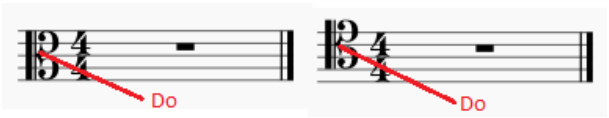

Figura 3.5: Clave de Do en 4ta. y 3ra. linea

A partir de estas claves se le puede poner nombre a las diferentes líneas y espacios del pentagrama, por ejemplo en la clave de sol en segunda línea, el pentagrama quedaría de la siguiente forma:

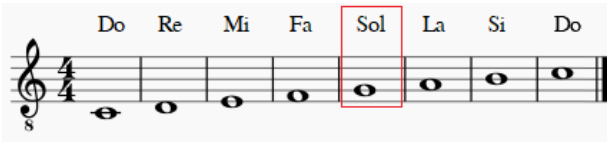

Figura 3.6: Nombre de las notas en un pentagrama con clave de Sol

3.1.3. Compases y tiempo

El compás se puede definir como la unidad métrica de la música, es la que nos dice que tiempo llevara la canción.

Existen una infinidad de compases los cuales podemos clasificar en dos grandes grupos:

- Compases regulares.
- Compases irregulares.

Los compases regulares están regidos por formas regulares las cuales dictan el tiempo, mientras que en los compases irregulares hay que determinar la base de tiempo de forma indirecta.

Cada nota puede tener un valor de tiempo y este valor estará determinado por el compás que se esté utilizando, por ejemplo en un compás de 4/4 las figuras musicales tendrán los siguientes valores:

Nombre	Figura	Silencio	Valor/Pulsos
Redonda	𝅝	𝄻	4 Tiempos
Blanca	𝅗𝅥	𝄼	2 Tiempos
Negra	♩	𝄽	1 Tiempo
Corchea	♪	𝄾	1/2 Tiempo
Semicorchea	𝅘𝅥𝅯	𝄿	1/4 Tiempo
Fusa	𝅘𝅥𝅰	𝅀	1/8 Tiempo

Tabla 3.1: Duración de notas musicales.

Como se puede observar para cada figura musical existe su silencio correspondiente, el cual tendrá el mismo valor solo que este caso no se producirá sonido alguno.

Podemos tener también los valores relativos de estas notas respecto a otras notas:

		𝅝	𝅗𝅥	♩	♪	𝅘𝅥𝅯	𝅘𝅥𝅰
Redonda	𝅝	1	2	4	8	16	32
Blanca	𝅗𝅥	1/2	1	2	4	8	16
Negra	♩	1/4	1/2	1	2	4	8
Corchea	♪	1/8	1/4	1/2	1	2	4
Semicorchea	𝅘𝅥𝅯	1/16	1/8	1/4	1/2	1	2
Fusa	𝅘𝅥𝅰	1/32	1/16	1/8	1/4	1/2	1

Tabla 3.2: Valor relativo de las notas.

De tal manera que en un compás tendremos dos datos los cuales se describen a continuación:

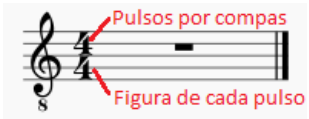

Figura 3.7: Compas

Se puede ver que este compás tendrá 2 pulsos y el valor de cada pulso será de un tiempo de negra, ya que el valor relativo de la negra respecto a la redonda es de 1/4.

Otro punto a considerar en los compases, es que estos tienen tiempos fuertes, semifuertes y débiles. La cuadratura de una armonía usa estos tiempos para indicar los cambios que se deben de hacer.

Este es un ejemplo de los tiempos en compases de 4/4, 3/4 y 2/4:

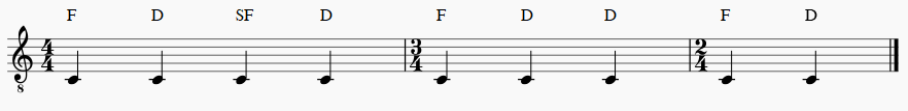

Figura 3.8: Tiempos fuertes y débiles

3.1.4. Escalas

Las escalas son una serie de notas musicales que siguen un orden establecido por intervalos desde una nota base. En el mundo de la música hay una infinidad de escalas pero todas parten de la escala mayor de Do:

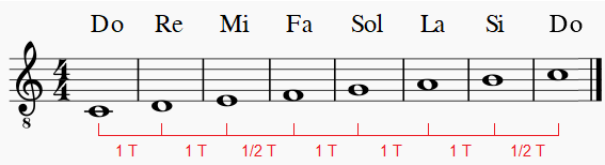

Figura 3.9: Escala de Do Mayor

En la escala mayor se tiene los siguientes intervalos: tono, tono, semitono, tono, tono, tono, semitono, si esta fórmula la aplicamos con las otras notas musicales podremos construir todas las escalas mayores.

Por ejemplo la escala de sol mayor quedaría de la siguiente manera:

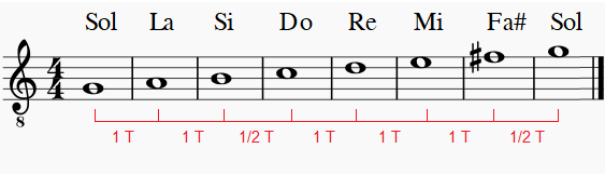

Figura 3.10: Escala de Sol Mayor

Para poder completar el tono completo de la fórmula de la escala mayor entre Mi y Fa se tuvo que poner una alteración en la nota de Fa.

La única escala mayor que no posee alteraciones es la escala mayor de Do, de ahí en más todas las demás escalas mayores tendrán al menos una alteración.

Las escalas menores parten de la escala mayor obteniendo su sexta nota y siguiendo las mismas notas de la escala mayor. Estas escalas menores se les conocen como relativas, ya que se basan en una escala mayor para su formación.

Por ejemplo la escala relativa de Do mayor seria La menor y esta escala posee exactamente las mismas notas que la escala mayor solamente que empieza desde la nota de La:

En este caso las notas son: La, Si, Do, Re, Mi, Fa, Sol, La.

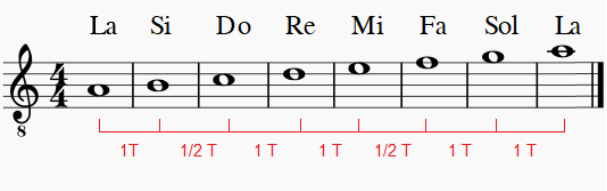

Figura 3.11: Escala de La menor

Esta relatividad puede ser usada con todas las escalas mayores para obtener sus relativas menores.

A pesar que las escalas mayores y menores poseen las mismas notas, no deben ser nunca confundidas ya que el sonido final producido es muy diferente, mientras las escalas mayores se usan para canciones se puede decir hasta cierto punto alegres, las escalas menores normalmente acompañan melodías melancólicas, esto no es en todos los casos.

3.1.5. Armaduras

Todas las alteraciones de una escala se pueden juntar al inicio del pentagrama para dar lugar a lo que se conoce como armaduras.

Estas armaduras indicaran que todas las notas que poseen alteraciones las mantendrán a lo largo de toda la canción.

Debido a que las escalas mayores y sus relativas menores poseen las mismas alteraciones comparten también la misma armadura:

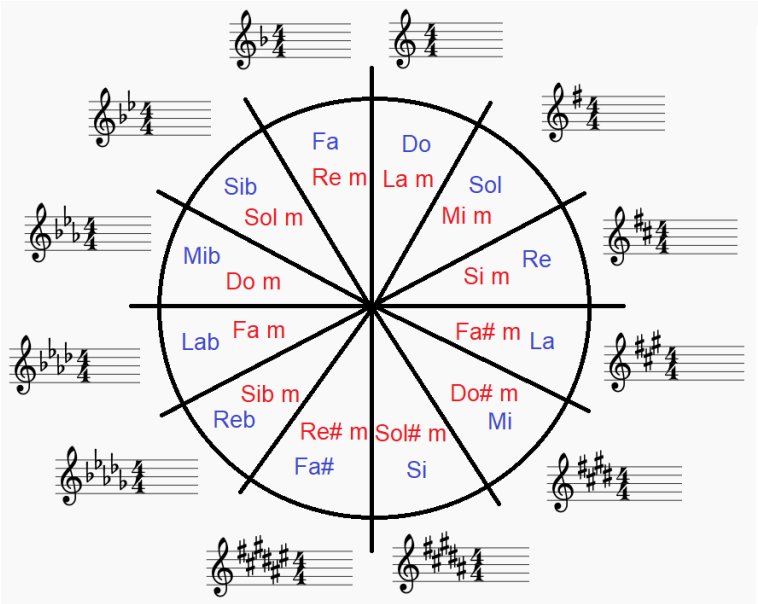

Figura 3.12: Armaduras

3.1.6. Tonalidades

La tonalidad de una canción se basa en la escala base de la canción y esta la podemos averiguar viendo la armadura que posee la canción, aunque como se vio anteriormente las escalas mayores y sus relativas poseen la misma armadura, así que antes de definir la tonalidad en base a la armadura también se debe de hacer un análisis de la interacción de las notas en la canción.

Dentro de las tonalidades tenemos mayores y menores, por ejemplo si vemos un pentagrama el cual no posee alteraciones podríamos asumir que la canción esta en Do mayor o en La menor, el siguiente paso sería ver la interacción de las notas en la canción para determinar correctamente la tonalidad.

3.1.7. Melodías y Armonías

Una melodía es una sucesión de notas de forma ascendente o descendente que llevan cierta cordura.

Las melodías suelen estar formadas por frases y generalmente se repiten a lo largo de una canción variando algunas notas intermedias. En este caso se trata de una sucesión de notas que no son tocadas al mismo tiempo sino que una nota es tocada después de la otra.

Las armonías son una conjunción de sonidos tocados al mismo tiempo, normalmente la sucesión de varios acordes armónicos está ligado directamente con la melodía de la canción.

La armonía ha cambiado considerablemente desde la época de la música barroca hasta la música moderna, anteriormente para generar sistemas armónicos se usaban varios instrumentos tocando una nota en específico y la conjunción de todos los sonidos daba como resultado la armonía, actualmente la armonía es creada a partir de acordes de instrumentos que puedan generar este tipo de condiciones.

Esta es una comparativa para diferenciar entre lo que sería una melodía y una armonía:

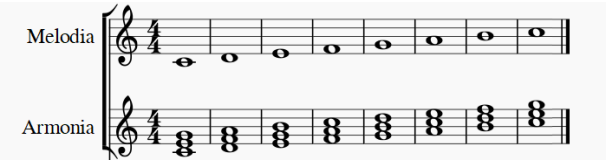

Figura 3.13: Melodía y Armonía

3.2. Formato MIDI

MIDI (Interfaz digital de instrumentos musicales) es un estándar técnico que describe un protocolo, una interfaz digital y conectores para la interoperabilidad entre varios instrumentos musicales electrónicos, software y dispositivos. [7]

Los archivos MIDI transmiten datos, no sonidos, y estos datos son almacenados en Tracks. Cada archivo MIDI puede tener 1 o más Tracks.

Los Tracks poseen mensajes de eventos que especifican información de notas (como tono y velocidad), así como señales de control para parámetros (como volumen, vibrato y señales de reloj). También hay eventos que dan información acerca del instrumento, clave musical y tiempo. En la figura 3.14 se observa la estructura de un archivo MIDI.

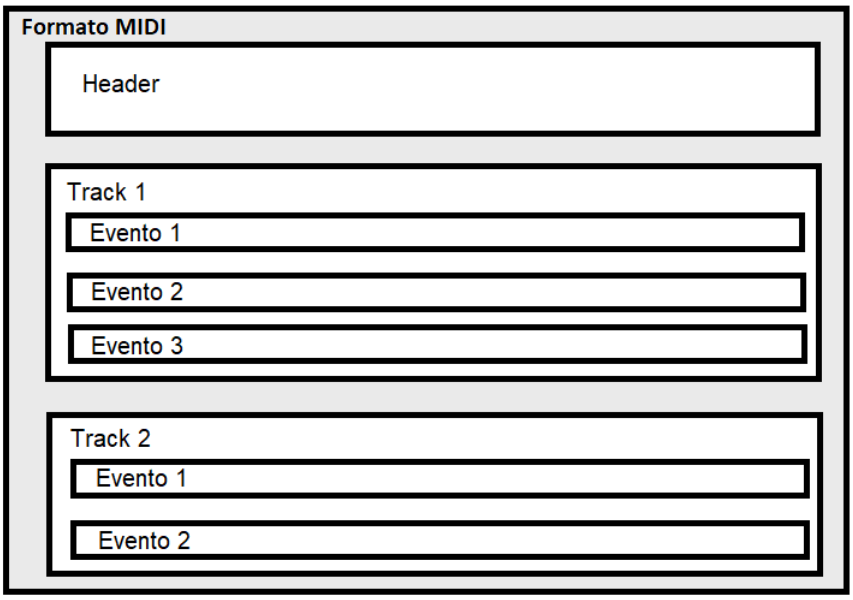

Figura 3.14: Formato MIDI

Los mensajes MIDI están estructurados en octetos, por lo que solo pueden existir 2^3 tipos de mensajes diferentes. Cuando se transmiten, a cada octeto se le agrega el bit de paridad con la finalidad de detectar errores en la transmisión o recepción.

En la tabla 3.3 se muestran los diferentes eventos que existen y sus parámetros:

Mensaje	Código	Parametro 1	Parametro 2
Note Off	0x8	Numero de nota	Velocidad
Note On	0x9	Numero de nota	Velocidad
Polyphonic aftertouch	0xA	Numero de nota	Presión
Control Change	0xB	Numero de controlador	Datos
Program Change	0xC	Numero de programa	-
Channel aftertouch	0xD	Presion	-
Pitch wheel	0xE	LSByte	MSByte
System Message	0xF	-	-

Tabla 3.3: Mensajes MIDI.

Cada uno de los mensajes se describen a continuación:

- **Note on.-** Este mensaje le indica al dispositivo, que debe iniciar una nota. Contiene la información de estado (número de canal, especificado por un entero dentro de [0 15] y dos valores de datos: un número de nota MIDI (el tono de la nota, un entero dentro de [0 127]) y una velocidad (que indica cómo la nota es reproducida, un entero dentro de [0 127]). Un ejemplo es "Note on, 0, 60, 50" que interpreta como: en el canal 1, comienza a reproducir un C medio con una velocidad de 50.

- **Note off.-** Indica que una nota termina. En esa situación, la velocidad indica qué tan rápido se libera la nota. Un ejemplo es "Note off, 0, 60, 20" que interpreta como: en el canal 1, deja de reproducir un C medio con una velocidad de 20. [7]

- **Polyphonic aftertouch.-** Algunos teclados de alta gama son capaces de detectar de forma permanente (decenas de veces por segundo) los cambios en la presión ejercida sobre cada una de sus teclas. En este caso, siempre que se produzca algún cambio, envían este mensaje.

- **Control Change.-** Este mensaje engloba 128 posibles mensajes de control diferentes Todos ellos afectan de alguna forma a la calidad del sonido; existen controles para modificar el volumen, la modulación, la reverberación, etc. [7]

- **Program Change.-** Evento para designar los diferentes sonidos disponibles en un sintetizador (instrumentos, efectos sonoros, etc.). [8]

- **Channel aftertouch.-** Este mensaje es parecido al Polyphonic aftertouch con la diferencia de que en lugar de enviar un mensaje de presión por cada nota, se envia uno por cada canal. [8]

- **Pitch wheel.-** la inmensa mayoría de teclados disponen a la izquierda, de dos pequeñas ruedas giratorias. Una de ellas (la que vuelve automáticamente a su posición central), se utiliza para desafinar ligeramente el sonido. Cuando la rueda gira, el teclado envía estos mensajes de forma continua (decenas de veces por segundo). [8]

- **System Message.-** Estos mensajes se comportan de forma diferente a todos los anteriores, ya que los cuatro bits restantes no indican un número de canal, y por ello, afectan globalmente al comportamiento de los dispositivos que los reciban. Estos cuatro bits restantes, definen de hecho dieciséis mensajes diferentes, que se reparten en tres grupos: los mensajes comunes de sistema, los mensajes de sistema de tiempo real y los mensajes de sistema exclusivo. [8]

En la siguiente tabla 3.4 se puede observar los valores de las notas y a que octava pertenecen:

Octava	C	C#	D	D#	E	F	F#	G	G#	A	A#	B
-1	0	1	2	3	4	5	6	7	8	9	10	11
0	12	13	14	15	16	17	18	19	20	21	22	23
1	24	25	26	27	28	29	30	31	32	33	34	35
2	36	37	38	39	40	41	42	43	44	45	46	47
3	48	49	50	51	52	53	54	55	56	57	58	59
4	60	61	62	63	64	65	66	67	68	69	70	71
5	72	73	74	75	76	77	78	79	80	81	82	83
6	84	85	86	87	88	89	90	91	92	93	94	95
7	96	97	98	99	100	101	102	103	104	105	106	107
8	108	109	110	111	112	113	114	115	116	117	118	119
9	120	121	122	123	124	125	126	127				

Tabla 3.4: Numero de nota en formato MIDI.

Cada evento de nota está realmente incrustado en un fragmento de pista, una estructura de datos que contiene un valor de tiempo delta que especifica la información

de temporización y el evento en sí. Un valor de tiempo delta representa la posición de tiempo, como un valor absoluto, del evento y podría representar:

- **Metrical time.-** Representa el número de pulsos desde el comienzo. Una referencia llamada división y es definida en el encabezado del archivo, especificando cuantos pulsos por nota de cuarto.

- **time-code-based time.-** Representa el tiempo relacionado con horas, minutos y segundos de la canción.

Un ejemplo de un extracto de un archivo MIDI y su partitura correspondiente es mostrado en las siguientes figuras 3.15 3.16.

```
TRACK 001 CHUNK
  Chunk length: 6816
       0     000017    Non-MIDI event, sequence/track name: Electric Guitar
       0     00002A    Program change, channel 01    Overdriven Guitar
       0     00002D    Pitch wheel control, channel 01    Value: 2000
       0     000031    Program change, channel 01    Overdriven Guitar
       0     000034    Non-MIDI event, tempo change: 342857 microseconds/quarter note
       0     00003B    Non-MIDI event, key signature: E Major
       0     000041    Non-MIDI event, time sig: 3/4 Notes/click: 24 Clock/quat: 4
       0     000049    Note on, channel 01    Note: E3   Velocity: 40
       0     00004D    Note on, channel 01    Note: G#3   Velocity: 40
       0     000051    Note on, channel 01    Note: B3   Velocity: 40
    3072     000056    Note off, channel 01    Note: E3   Release velocity: 00
       0     00005A    Note off, channel 01    Note: G#3   Release velocity: 00
       0     00005E    Note off, channel 01    Note: B3   Release velocity: 00
       0     000062    Note on, channel 01    Note: E3   Velocity: 40
       0     000066    Note on, channel 01    Note: A3   Velocity: 40
       0     00006A    Note on, channel 01    Note: B3   Velocity: 40
     512     00006F    Note off, channel 01    Note: E3   Release velocity: 00
       0     000073    Note off, channel 01    Note: A3   Release velocity: 00
       0     000077    Note off, channel 01    Note: B3   Release velocity: 00
     512     00007C    Note on, channel 01    Note: E3   Velocity: 40
       0     000080    Note on, channel 01    Note: A3   Velocity: 40
       0     000084    Note on, channel 01    Note: B3   Velocity: 40
```

Figura 3.15: Extracto de un formato MIDI

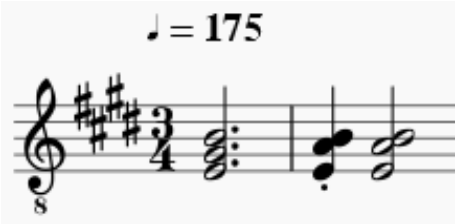

Figura 3.16: Partitura correspondiente al extracto MIDI

Dentro del formato MIDI se pueden representar en cada Track un instrumento diferente, para realizar esto se utiliza el evento "Program Change", el cual dependiendo de su valor, será el instrumento a reproducir.

PC	Instrumento	PC	Instrumento	PC	Instrumento	PC	Instrumento
1	Acoustic Grand Piano	9	Celesta	17	Drawbar Organ	25	Acoustic Guitar (nylon)
2	Bright Acoustic Piano	10	Glockenspiel	18	Percussive Organ	26	Acoustic Guitar (steel)
3	Electric Grand Piano	11	Music Box	19	Rock Organ	27	Electric Guitar (jazz)
4	Honky-tonk Piano	12	Vibraphone	20	Church Organ	28	Electric Guitar (clean)
5	Electric Piano 1	13	Marimba	21	Reed Organ	29	Electric Guitar (muted)
6	Electric Piano 2	14	Xylophone	22	Accordion	30	Overdriven Guitar
7	Harpsichord	15	Tubular Bells	23	Harmonica	31	Distortion Guitar
8	Clavi	16	Dulcimer	24	Tango Accordion	32	Guitar harmonics
33	Acoustic Bass	41	Violin	49	String Ensemble 1	57	Trumpet
34	Electric Bass (finger)	42	Viola	50	String Ensemble 2	58	Trombone
35	Electric Bass (pick)	43	Cello	51	SynthStrings 1	59	Tuba
36	Fretless Bass	44	Contrabass	52	SynthStrings 2	60	Muted Trumpet
37	Slap Bass 1	45	Tremolo Strings	53	Choir Aahs	61	French Horn
38	Slap Bass 2	46	Pizzicato Strings	54	Voice Oohs	62	Brass Section
39	Synth Bass 1	47	Orchestral Harp	55	Synth Voice	63	SynthBrass 1
40	Synth Bass 2	48	Timpani	56	Orchestra Hit	64	SynthBrass 2
65	Soprano Sax	73	Piccolo	81	Lead 1 (square)	89	Pad 1 (new age)
66	Alto Sax	74	Flute	82	Lead 2 (sawtooth)	90	Pad 2 (warm)
67	Tenor Sax	75	Recorder	83	Lead 3 (calliope)	91	Pad 3 (polysynth)
68	Baritone Sax	76	Pan Flute	84	Lead 4 (chiff)	92	Pad 4 (choir)
69	Oboe	77	Blown Bottle	85	Lead 5 (charang)	93	Pad 5 (bowed)
70	English Horn	78	Shakuhachi	86	Lead 6 (voice)	94	Pad 6 (metallic)
71	Bassoon	79	Whistle	87	Lead 7 (fifths)	95	Pad 7 (halo)
72	Clarinet	80	Ocarina	88	Lead 8 (bass + lead)	96	Pad 8 (sweep)
97	FX 1 (rain)	105	Sitar	113	Tinkle Bell	121	Guitar Fret Noise
98	FX 2 (soundtrack)	106	Banjo	114	Agogo	122	Breath Noise
99	FX 3 (crystal)	107	Shamisen	115	Steel Drums	123	Seashore
100	FX 4 (atmosphere)	108	Koto	116	Woodblock	124	Bird Tweet
101	FX 5 (brightness)	109	Kalimba	117	Taiko Drum	125	Telephone Ring
102	FX 6 (goblins)	110	Bag pipe	118	Melodic Tom	126	Helicopter
103	FX 7 (echoes)	111	Fiddle	119	Synth Drum	127	Applause
104	FX 8 (sci-fi)	112	Shanai	120	Reverse Cymbal	128	Gunshot

Tabla 3.5: Instrumentos en formato MIDI.

Los instrumentos de percusión no usan este evento, en lugar de eso usan el canal 10 para transmitir los datos.

3.3. Algoritmo Krumhansl-Schmuckler

Este algoritmo es usado para determinar o asignar la tonalidad más apropiada a una serie de notas musicales. Fue diseñado y desarrollado por Carol L. Krumhansl y Mark A. Schmuckler.

Para determinar cuál es la tonalidad más apropiada, el algoritmo hace uso del coeficiente de correlación de Pearson:

$$R = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2 \sum_{i=1}^{n}(y_i - \bar{y})^2}} \tag{3.3.1}$$

El cual dado un conjunto de datos (x, y) determina un valor entre $[-1, 1]$ que indica si el conjunto de datos x tiene relación con el conjunto de datos y:

- **0 :** Indica que no hay relación lineal entre x y y
- **1 :** Indica que hay una correlación perfecta positiva, si x crece y también.
- **-1 :** Indica que hay una correlación perfecta negativa, si x crece y decrece.

En el algoritmo para encontrar la tonalidad, una de las coordenadas representa un perfil de una tonalidad mayor (tabla 3.6) o una tonalidad menor (tabla 3.7).

Do	Do#	Re	Re#	Mi	Fa	Fa#	Sol	Sol#	La	La#	Si
6.35	2.23	3.48	2.33	4.38	4.09	2.52	5.19	2.39	3.66	2.29	2.88

Tabla 3.6: Perfil de tonalidades mayores.

la	la#	si	do	do#	re	re#	mi	fa	fa#	sol	sol#
6.33	2.68	3.52	5.38	2.60	3.53	2.54	4.75	3.98	2.69	3.34	3.17

Tabla 3.7: Perfil de tonalidades menores.

El otro valor de las coordenadas es el total de duración de cada nota en la pieza musical siendo analizada. En la tabla 3.8 se observa un ejemplo de los valores de una canción:

C	C#	D	D#	E	F	F#	G	G#	A	A#	B
432	231	0	405	12	316	4	126	612	0	191	1

Tabla 3.8: Ejemplo de duración de notas en una canción.

El algoritmo calcula el coeficiente de correlación para todas las posibles tonalidades mayores y menores. En las tablas 3.9 y 3.10 se muestra un ejemplo de cómo se empiezan a relacionar los datos de la canción con los perfiles de las tonalidades:

(Do, C)	(6.35, 432)
(Do#, C#)	(2.23, 231)
(Re, D)	(3.48, 0)
(Re#, D#)	(2.33, 405)
(Mi, E)	(4.38, 12)
(Fa, F)	(4.09, 316)
(Fa#, F#)	(2.52, 4)
(Sol, G)	(5.19, 126)
(Sol#, G#)	(2.39, 612)
(La, A)	(3.66, 0)
(La#, A#)	(2.29, 191)
(Si, B)	(2.88, 1)

Tabla 3.9: Correlación de Do Mayor.

En el caso de la tonalidad de Do Mayor obtenemos un coeficiente de correlación de $R = 0.00009$, lo que significa que no hay probabilidad de que esta sea la tonalidad adecuada para la canción.

Si se grafican los puntos de las coordenadas (x, y) para esta tonalidad se verá que los puntos están demasiados dispersos entre sí, por lo tanto no podemos encontrar una línea recta que prediga la posición de nuevos puntos.

Ahora vamos a analizar qué pasa con la tonalidad de Re Mayor.

(Do, D)	(6.35, 0)
(Do#, D#)	(2.23, 405)
(Re, E)	(3.48, 12)
(Re#, F)	(2.33, 316)
(Mi, F#)	(4.38, 4)
(Fa, G)	(4.09, 126)
(Fa#, G#)	(2.52, 612)
(Sol, A)	(5.19, 0)
(Sol#, A#)	(2.39, 191)
(La, B)	(3.66, 1)
(La#, C)	(2.29, 432)
(Si, C#)	(2.88, 231)

Tabla 3.10: Correlación de Re Mayor.

En este caso los puntos no se encuentran tan dispersos, por lo que obtenemos un coeficiente de correlación de $R = -0.74$, lo que significa que existe una gran correlación negativa.

Si se realiza este mismo ejercicio con todas las posibles tonalidades, obtendremos que la tonalidad que obtuvo mayor correlación es Sol# Mayor con un coeficiente de $R = 0.97$, por lo tanto esta sería la tonalidad más adecuada de la pieza.

Este algoritmo regresa una lista ordenada de todos los valores posibles y su coeficiente de correlación, donde el primer elemento es el que mayor correlación contiene.

Capítulo 4

Desarrollo

En este proyecto se busca que las redes neuronales sean capaces de aprender las diferentes tonalidades de las canciones, y que puedan crear nuevas composiciones que tengan sentido musicalmente hablando, tomando como base una canción de entrada en cierta tonalidad.

Esto será de gran ayuda para un músico ya que podrá introducir sus propias canciones y en base a las notas de sus canciones, la red será capaz de generar música completamente nueva en la misma tonalidad de la canción de entrada. Estas canciones en definitiva no son composiciones finales, el músico podrá identificar las frases que sean de su agrado y continuar con la composición.

4.1. Datos

La base de datos de archivos MIDI que se utilizó en este proyecto se llama "Clean MIDI subset" la cual se usó en la tesis doctoral "Learning-Based Methods for Comparing Sequences, with Applications to Audio-to-MIDI Alignment and Matching" [7]. Esta base de datos contiene más de 17,000 canciones en formato MIDI del género Rock y Pop, por lo que la red utilizada en este proyecto fue entrenada para asimilar este tipo de estilos.

Para el procesamiento de los archivos MIDI se utilizó una librería de software libre llamada Music21, la cual nos permite de una manera fácil trabajar con estos archivos, sin embargo se realizó una adaptación para que funcionara correctamente con diferentes sonidos de guitarra y bajo eléctrico. Esta librería es compatible con Python3 por lo que nuestro programa está hecho en este lenguaje de programación.

Los archivos MIDI de esta base de datos poseen varios Tracks, cada uno con un

instrumento diferente, de los cuales en este proyecto solamente se tomaran en cuenta los Tracks de guitarra y bajo eléctrico, los demás Tracks serán ignorados.

Cada canción tiene una duración promedio de 3 minutos, por lo que la cantidad de notas que procesa la red es bastante considerable.

4.2. Arquitectura de la aplicación

Este proyecto posee varios módulos, cada uno funciona independientemente de los otros, sin embargo existen interconexiones entre ellos para el paso de información, esto nos permite tener una buena modulación del código y realizar cambios en un módulo sin afectar a los otros.

Estos módulos se pueden observar en la siguiente figura:

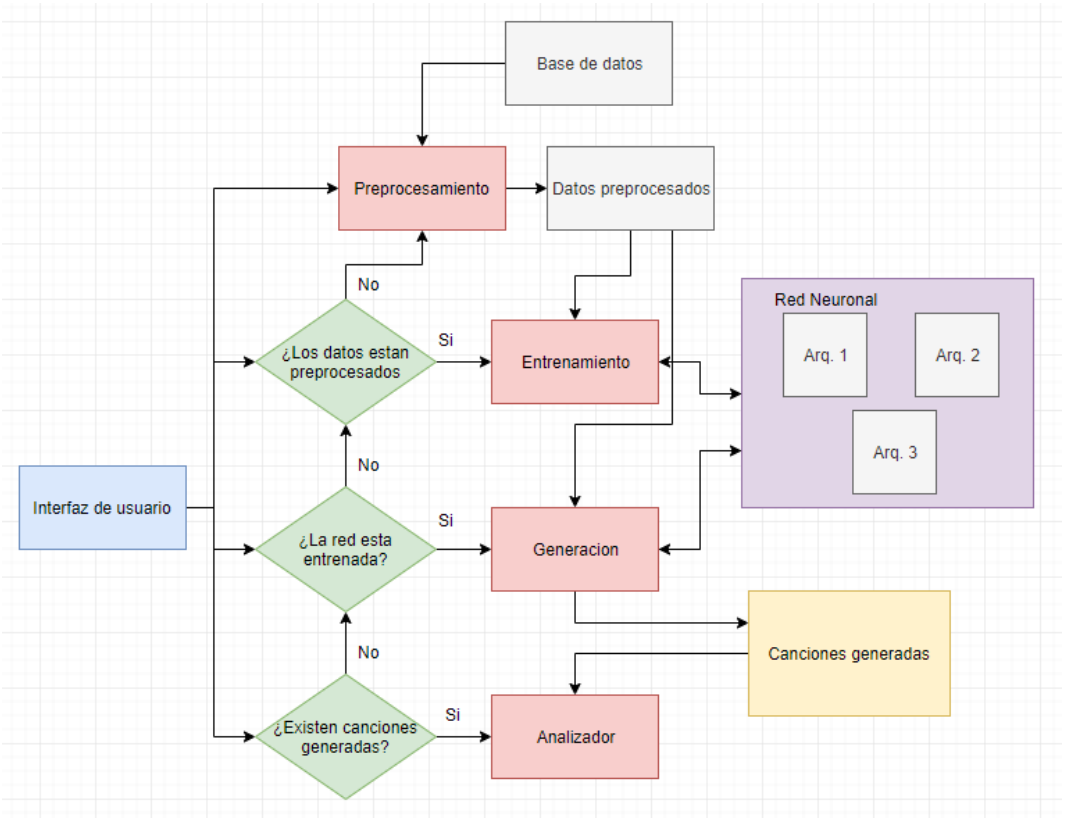

Figura 4.1: Arquitectura de la aplicación

Dentro de esta arquitectura tenemos módulos activos y módulos pasivos. Los módulos activos son aquellos que reciben y procesan información, mientras que los

elementos pasivos solamente son para almacenar o consultar información.

Los elementos activos que se observan son:

- UI.
- Preprocesamiento.
- Red neuronal.
- Entrenamiento.
- Generación.
- Analizador.

Mientras que los elementos pasivos son:

- Base de datos.
- Datos preprocesados.
- Canciones generadas.

A continuación se describirán cada módulo del sistema.

4.2.1. UI

Este módulo es una pequeña interfaz de usuario de consola basada en menus, la cual nos permite accesar a los otros módulos del sistema.

El menú principal contiene el acceso a los siguientes módulos:

1.- Preprocesamiento.
2.- Entrenamiento.
3.- Generación.
4.- Analizador.

Como se puede observar en la figura 4.1 hay algunos módulos que para poderse ejecutar es necesario correr otro módulo primero, con el fin de obtener la información necesaria para este módulo. El único módulo que no tiene condicionantes es el de Preprocesamiento, por lo tanto es el primer módulo que se debería de ejecutar en el programa.

4.2.2. Preprocesamiento

En este módulo se realiza el preprocesamiento de la base de datos, esto es con el fin de extraer únicamente los Tracks de guitarra y bajo de los archivos MIDI.

Como se mencionó anteriormente esta base de datos tiene archivos de música pop y rock, sin embargo, no todos los archivos son válidos para ser procesados por este programa, debido a que algunos no contienen información completa en sus Tracks para su correcto procesamiento usando la librería Music21.

La librería Music21 está enfocada principalmente para hacer reconocimiento de pianos y sus derivados, por lo tanto se tuvo que realizar un reacondicionamiento de esta librería para el correcto funcionamiento con los instrumentos de guitarra y bajo.

El modo en que se puede reconocer si un track es de guitarra o de bajo es en base a un evento llamado "Program Change", este evento no lo poseen todos los Tracks MIDI por lo que en ocasiones es complicado determinar cuál es el instrumento que se está ejecutando, en estos casos el sintetizador MIDI es el encargado de asignarle un instrumento por defecto, el cual generalmente es el piano.

Estos eventos de "Program Change" están agrupados en familias, de esta manera es más fácil su identificación, a nosotros únicamente nos interesa la familia de guitarra y bajo, por lo tanto los únicos "Program Change"que estaremos buscando son:

Program Change	Familia
25-32	Guitarra
33-40	Bajo

Tabla 4.1: Familias de guitarra y bajo.

Este elemento de "Program Change" no es una propiedad inherente de los Tracks, por lo que hay que hacer una búsqueda en todos los eventos de cada Track.

Normalmente encontraremos varios Tracks que contengan sonidos de guitarra o bajo en un archivo MIDI y esto se debe que para tocar por ejemplo una guitarra con distorsión se utiliza un Track, y para un sonido limpio se utiliza otro Track. También existen casos en los cuales aun y cuando se trate del mismo sonido las notas están esparcidas en varios Tracks, esto depende de cómo fue que se creó el archivo MIDI.

Es por estas circunstancias la importancia de tener este módulo, ya que no solamente se encarga de separar los Tracks de guitarra y bajo en archivos separados,

sino que también valida la integridad de los archivos MIDI y descarta los Tracks que tienen muchos silencios y muy pocas notas activas.

Para tener un procesamiento más rápido a la hora del proceso de entrenamiento o generación musical, se realiza la exportación de las notas y silencios a un archivo de manera serializada, de esta manera nos evitamos estar procesando los archivos MIDI y solamente deserializamos el archivo exportado.

Como se observa en la figura 4.1 la salida de este módulo serán los archivos preprocesados los cuales son necesarios para realizar la parte de entrenamiento.

4.2.3. Red neuronal

En este proyecto se están utilizando 3 tipos de arquitecturas diferentes para validar cuál de las 3 arroja mejores resultados:

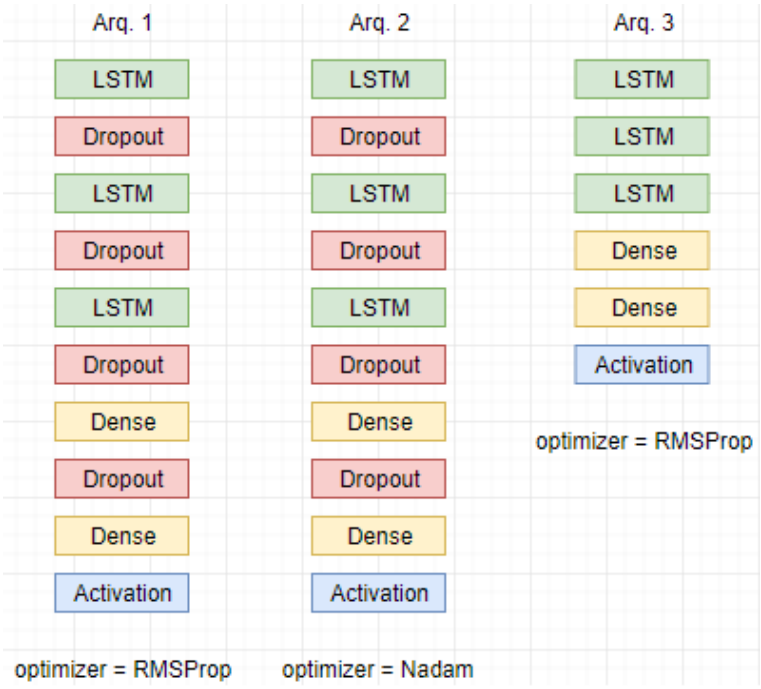

Figura 4.2: Arquitectura de las redes usadas

Como se puede observar las 3 redes poseen 3 capas de LSTM sin embargo en las 2 primeras se están usando unas capas de Dropout intermedias, estas capas realizan la función de poner en 0 aleatoriamente algunas entradas para prevenir el sobre entrenamiento, sin embargo en la última arquitectura, se eliminaron estas capas para verificar el comportamiento que podría tener una red así.

Otro de los aspectos a verificar es el mejor optimizador, en este caso se está usando un optimizador RMSProp en la primera y tercera arquitectura. Este optimizador funciona calculando dividiendo la tasa de aprendizaje por un promedio decreciente exponencial de gradientes cuadrados:

$$E[g^2]_t = 0.9E[g^2]_{t-1} + 0.1g_t^2 \tag{4.2.1}$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t \tag{4.2.2}$$

En la figura 4.2 se observa que la primera y segunda red son prácticamente iguales, lo único que las diferencia es que la segunda usa un optimizador Nadam, la cual es una combinación de RMSProp, Momentum y Nesterov:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon}(\beta_1 \hat{m}_t + \frac{(1-\beta_1)g_t}{1-\beta_1^t}) \tag{4.2.3}$$

Las últimas capas de las redes son densas las cuales interconectan todas las neuronas de una capa con otra, esto nos permite mejorar la tasa de aprendizaje.

Finalmente la capa de activación es por medio de una función softmax o función exponencial normalizada, la cual es una generalización de la función logística:

$$\sigma(z)_j = \frac{e^{z_t}}{\sum_{k=1}^{K} e^{z_k}} \quad \text{para } j = 1, ..., K \tag{4.2.4}$$

4.2.4. Entrenamiento

En este módulo se realiza toda la parte del entrenamiento. Es necesario seleccionar el tipo de arquitectura que se usara para el entrenamiento, así como también el instrumento a entrenar, es importante mencionar que cada instrumento es entrenado por separado.

Se toman los datos preprocesados (archivos exportados con las notas) los cuales son deserializados para obtener un arreglo únicamente con las notas y silencios, con este arreglo se empiezan a generar secuencias de 100 notas y silencios. Estas secuencias son insertadas en forma de matriz a una función de mapeo, la cual les asignara un valor único a cada nota, esto es debido que la red neuronal solamente reconoce números.

Finalmente esta matriz es normalizada para poder ser procesada por la red LSTM.

En la siguiente figura podemos observar como es el flujo de entrenamiento:

Figura 4.3: Flujo de entrenamiento

En este proyecto las diferentes arquitecturas fueron entrenadas con 200 épocas para cada instrumento, obteniendo buenos resultados de salida. Es posible entrenar usando más épocas sin embargo el poder de procesamiento nos limita un poco a esto, ya que para poder entrenar un instrumento lleva aproximadamente 2 semanas y media por cada red. Se está usando un servidor de Amazon con tarjetas NVIDIA para realizar todo el proceso de entrenamiento y aun así el tiempo de entrenamiento es bastante considerable.

En cada época se verifica si la red está teniendo una mejora en su aprendizaje, si es así se guarda un archivo con todos los pesos de las neuronas, con esto nos aseguramos de siempre guardar los mejores valores de pesos.

Además de este archivo de pesos, también se guarda un histórico con los valores de aprendizaje y de disminución del error, para poder graficarlo después y ver cómo fue evolucionando nuestra red.

De todo el conjunto de entrenamiento se usa un 75 % para la fase de entrenamiento y el otro 25 % para la fase de validación, de esta manera tenemos un gran conjunto de validación.

Los datos son introducidos a la red en forma de lotes, cada uno de tamaño 64 y la actualización de los pesos de la red se realiza después de que cada lote sea procesado.

4.2.5. Generación

Este es el módulo encargado de la generación de nuevas canciones en base a una canción de entrada, y una red entrenada para guitarra y bajo.

El usuario puede elegir con cual arquitectura realizar la generación musical, sin embargo, es importante mencionar que si no se tiene entrenada la arquitectura tanto

para guitarra como para bajo no podrá ser usada.

La forma en que funciona este módulo es tomando una canción como base, la cual puede ser de la base de datos preprocesados, o una composición propia del usuario, se extraerán los datos de esta canción para obtener un arreglo de notas y silencios, con el cual se generaran secuencias de 100 notas y silencios, tal como en el caso del módulo de entrenamiento son puestas en una matriz para posteriormente pasarlas por una función de mapeo.

Las secuencias son insertadas a la red una por una y usando la funcionalidad de predicción de la red, se obtiene una secuencia de salida con las ponderaciones de la proximidad a la siguiente nota. Esta secuencia se analiza y se toma el número mayor para descubrir cuál es la siguiente nota generada. Este proceso se repite hasta tener el número de notas deseadas por el usuario.

En esta figura se puede visualizar más fácil este procedimiento:

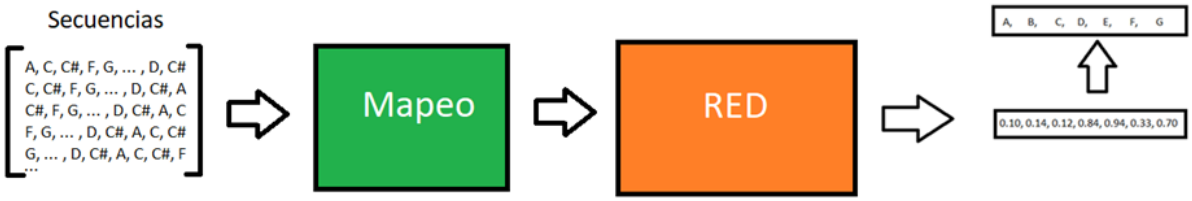

Figura 4.4: Flujo de generación de notas

El módulo generara la salida de guitarra o de bajo según sea el caso, para posteriormente crear un archivo en formato MIDI que contendrá estas notas.

La canción de salida deberá tener la misma tonalidad de la canción base, esto no siempre se puede cumplir ya que la red no tiene un 100 % de eficacia, sin embargo, los resultados obtenidos en este proyecto fueron bastante alentadores llegando a un máximo de 75 % aproximadamente.

Las canciones son guardadas con un número genérico seguido de una numeración, esto es para poder generar muchas canciones sin preocuparnos de que se puedan sobrescribir canciones anteriormente creadas.

4.2.6. Analizador

Finalmente el módulo de analizador como su nombre lo dice es el encargado de analizar las canciones generadas por el programa. Si no se tienen canciones generadas

por el programa este módulo no podrá ser usado.

Este módulo contiene 3 utilerias:

- Visor.
- Reproductor.
- Analizador.

El visor nos permite visualizar una canción en forma de tablatura en el programa que tengamos configurado por defecto en Music21, esto nos permite analizar las alturas y los intervalos entre las notas, así como también visualizar las secuencias generadas de guitarra y bajo.

Es posible realizar modificaciones a las canciones creadas usando el visor, y verificar como se escucha con estas modificaciones. En este caso se tiene configurado MuseScore con Music21, el cual es un programa gratis para crear, reproducir e imprimir partituras.

El reproductor por defecto instalado en Windows nos permite reproducir las canciones generadas. Es importante mencionar que cada reproductor tiene su propio sonido a la hora de reproducir los archivos MIDI.

El analizador utiliza el algoritmo de Krumhansl-Schmuckler para realizar el cálculo de la tonalidad de la canción generada. Este algoritmo retorna un arreglo con las posibles tonalidades de la canción, siendo el primer elemento la tonalidad con mayor probabilidad.

Para el cálculo de la tonalidad se toma en cuenta las notas de las escalas para las diferentes tonalidades. Como se vio en el capítulo de Teoría musical, cada tonalidad tiene sus propias alteraciones, sin embargo existen tonalidades relativas, las cuales poseen exactamente las mismas alteraciones, por lo que en ocasiones este algoritmo determina que la tonalidad de la canción es una tonalidad relativa, en este caso tomamos como un cálculo correcto, ya que para determinar más a fondo si es una tonalidad relativa o no, es necesario analizar las notas dominantes de las canciones, así como las funciones tonales.

Una vez obtenida la tonalidad con este algoritmo, se compara contra la tonalidad con la que fue creada esta canción, si son iguales, lo catalogamos como un caso exitoso, de esta manera podemos verificar si nuestra red en realidad está aprendiendo a crear canciones con diferentes tonalidades.

Capítulo 5

Resultados

En esta sección se presentaran los resultados obtenidos de los diferentes experimentos que se realizaron en este proyecto. Estos experimentos se dividieron en 3 secciones:

- Etapa de entrenamiento.
- Etapa de generación.
- Etapa de validación.

En cada etapa se intenta evaluar a los diferentes instrumentos de la manera más objetiva posible.

5.1. Etapa de entrenamiento

En la etapa de entrenamiento se usaron las 3 diferentes arquitecturas de redes neuronales de manera independiente, es decir, primeramente se entrenó con una arquitectura y se guardaron los mejores valores de los pesos, para posteriormente realizar el mismo procedimiento con las otras arquitecturas.

Este entrenamiento fue realizado usando toda la base de datos preprocesados, los cuales tomaron en cuenta únicamente los archivos MIDI que contaran con al menos un Track de guitarra y uno de bajo.

Esta base de datos fue segmentada en 2 partes, dejando el 75 % de los datos para entrenamiento y el restante 25 % para la validación.

Las redes se entrenaron durante 200 épocas para cada instrumento, es decir, primero se entrenó la red para que generara patrones de guitarra y luego se entrenó para bajo, por lo tanto se tienen resultados de los 2 procesos.

5.1.1. Arquitectura 1

Esta arquitectura posee 3 capas de redes LSTM con capas de Dropout intermedias, así como también unas capas Densas al final y un optimizador RMSProp.

5.1.1.1. Guitarra

En la Figura 5.1 se muestra como fue disminuyendo el error usando esta arquitectura para guitarra.

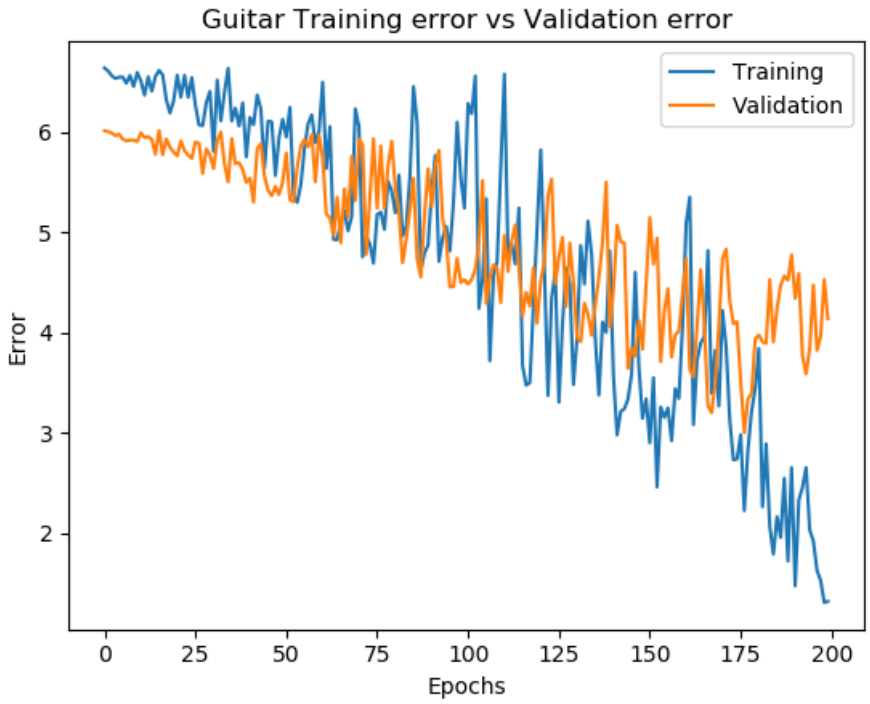

Figura 5.1: Gráfica de error para guitarra en arquitectura 1

En la Tabla 5.1 se puede observar que la tendencia del error es a la baja, llegando a valores de 1.3161 en la fase de entrenamiento y de 4.1383 en la fase de validación, y aunque por la época 100 pareciera que el error va a tender a subir, no es así al final.

Fase	Valor inicial	Valor final	Valor mínimo	Valor máximo
Entrenamiento	6.6410	1.3161	1.3055	6.6410
Validación	6.0144	4.1383	3.0009	6.0167

Tabla 5.1: Valores de error para guitarra en arquitectura 1.

En la Figura 5.2 se muestra como fue aprendiendo la red con esta arquitectura para guitarra.

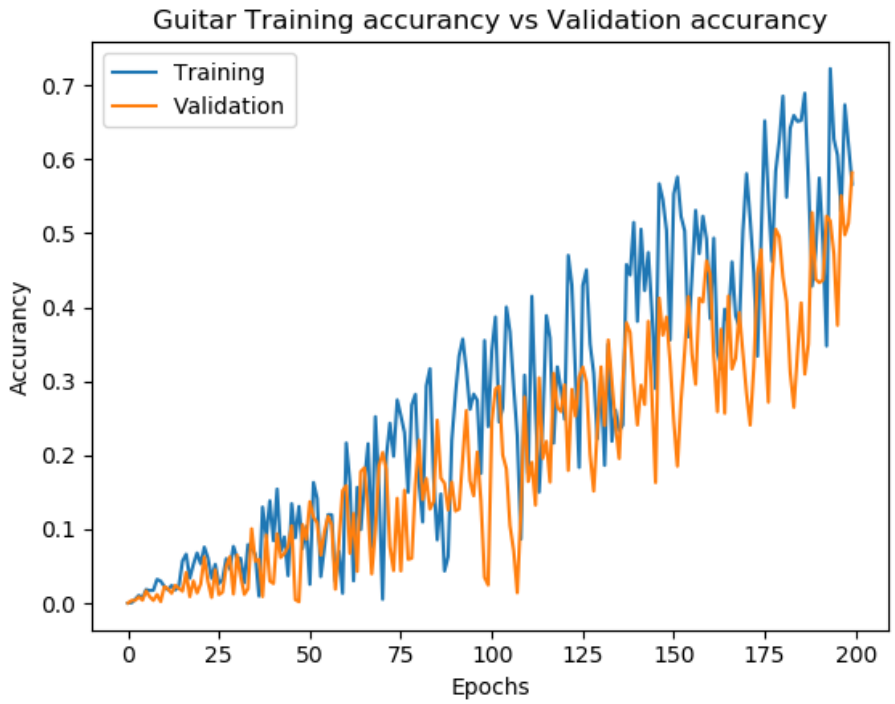

Figura 5.2: Gráfica de aprendizaje para guitarra en arquitectura 1

En la Tabla 5.2 se observa que para la fase de entrenamiento se llegó a un valor final de 56.66 % y en la fase de validación a un 58.18 %, este porcentaje podría parecer bajo, sin embargo se tuvieron máximos en algunas épocas de hasta 72.26 % y 58.18 % respectivamente.

El resultado final de esta arquitectura es bastante bueno, ya que la fase de validación se comportó de manera muy lineal en comparación de la fase de entrenamiento, obteniendo un valor muy parecido al final.

Fase	Valor inicial	Valor final	Valor mínimo	Valor máximo
Entrenamiento	0.0001	0.5666	0.0001	0.7226
Validación	0.0001	0.5818	0.0001	0.5818

Tabla 5.2: Valores de aprendizaje para guitarra en arquitectura 1.

5.1.1.2. Bajo

En la Figura 5.3 se muestra como fue disminuyendo el error usando esta arquitectura para bajo.

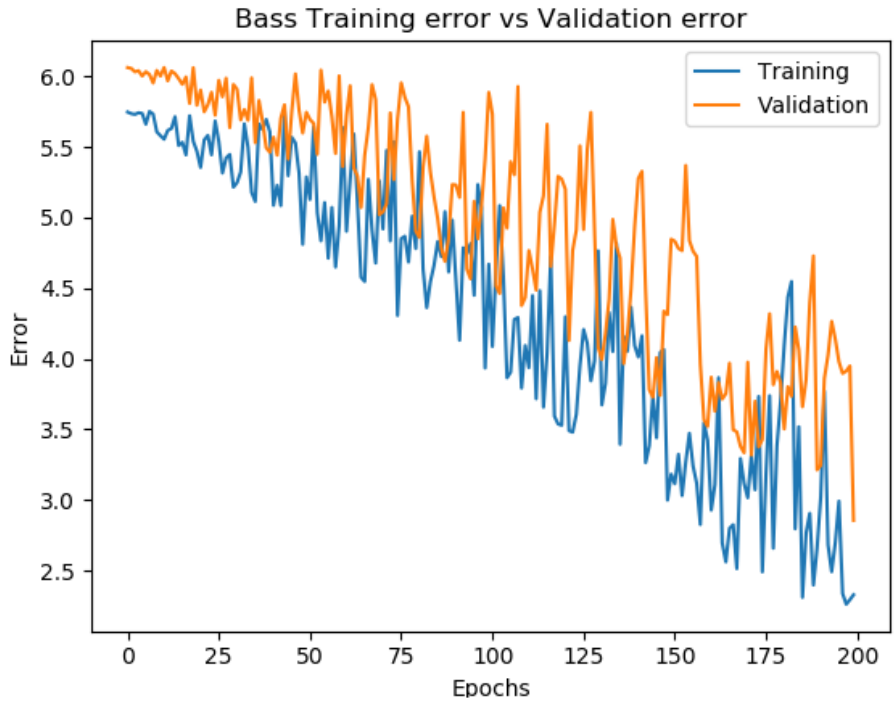

Figura 5.3: Gráfica de error para bajo en arquitectura 1

En la Tabla 5.3 se puede observar que el error llegó a valores de 2.3281 en la fase de entrenamiento y de 2.8547 en la fase de validación.

Después de la época 150 parece que el error va a volver a subir tanto en la fase de entrenamiento como en la de validación, sin embargo al pasar las épocas se recupera la tendencia a la baja.

Fase	Valor inicial	Valor final	Valor mínimo	Valor máximo
Entrenamiento	5.7464	2.3281	2.2614	5.7531
Validación	6.0615	2.8547	2.8547	6.0620

Tabla 5.3: Valores de error para bajo en arquitectura 1.

En la Figura 5.4 se muestra como fue aprendiendo la red con esta arquitectura para bajo.

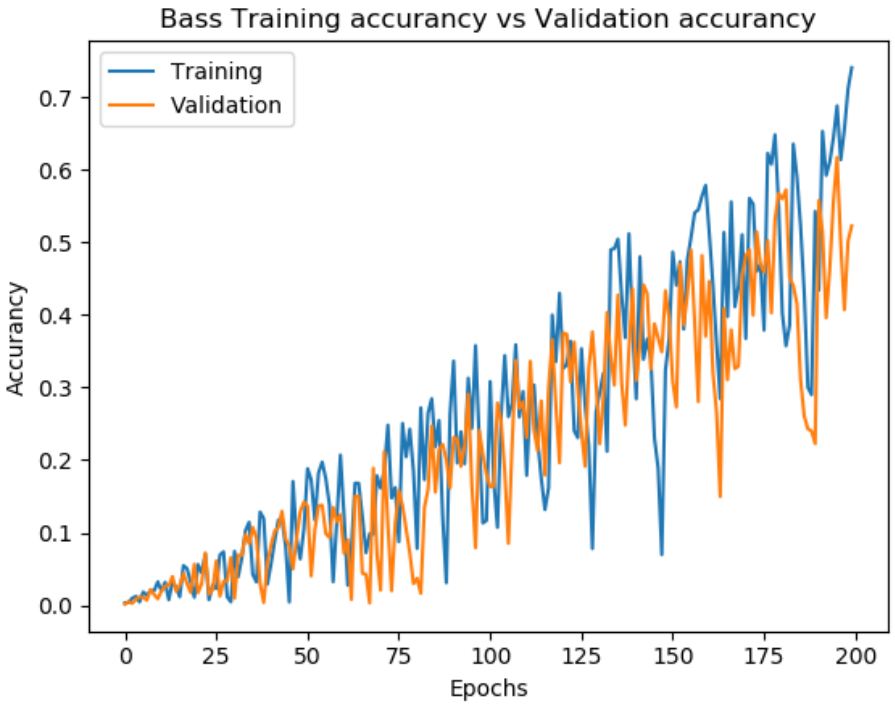

Figura 5.4: Gráfica de aprendizaje para bajo en arquitectura 1

En la Tabla 5.4 se observa que para la fase de entrenamiento se llegó a un valor final de 74.08 % y en la fase de validación a un 52.28 %.

Al igual que la guitarra, el bajo tuvo un excelente aprendizaje con esta arquitectura, a pesar de que tuvo algunas caídas bastante pronunciadas, cerca de la época 150.

El valor máximo obtenido es bastante similar al valor máximo para la guitarra con esta misma arquitectura.

Fase	Valor inicial	Valor final	Valor mínimo	Valor máximo
Entrenamiento	0.0034	0.7408	0.0034	0.7408
Validación	0.0014	0.5228	0.0014	0.6170

Tabla 5.4: Valores de aprendizaje para bajo en arquitectura 1.

5.1.2. Arquitectura 2

Esta arquitectura es muy similar a la numero 1, posee las mismas capas de LSTM, Dropout y Densas, sin embargo esta arquitectura usa un optimizador Nadam.

5.1.2.1. Guitarra

En la Figura 5.5 se muestra como fue disminuyendo el error usando esta arquitectura para guitarra. Se puede observar que en la fase de validación el error disminuyo más lentamente que en la fase de entrenamiento.

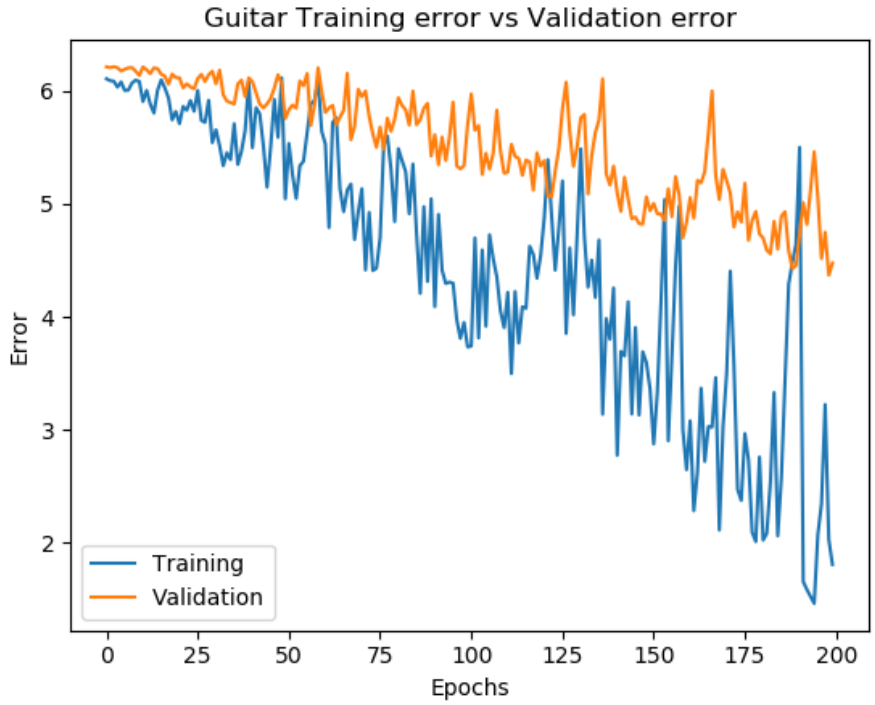

Figura 5.5: Gráfica de error para guitarra en arquitectura 2

En la Tabla 5.5 se observa que el error disminuyo a niveles de 1.8103 y 4.4766 en las fases de entrenamiento y validación respectivamente.

Fase	Valor inicial	Valor final	Valor mínimo	Valor máximo
Entrenamiento	6.1054	1.8103	1.4658	6.1138
Validación	6.2114	4.4766	4.3697	6.2126

Tabla 5.5: Valores de error para guitarra en arquitectura 2.

En la Figura 5.6 se muestra como fue aprendiendo la red con esta arquitectura para guitarra. Se tuvieron grandes caídas en la fase de validación a partir de la época 150.

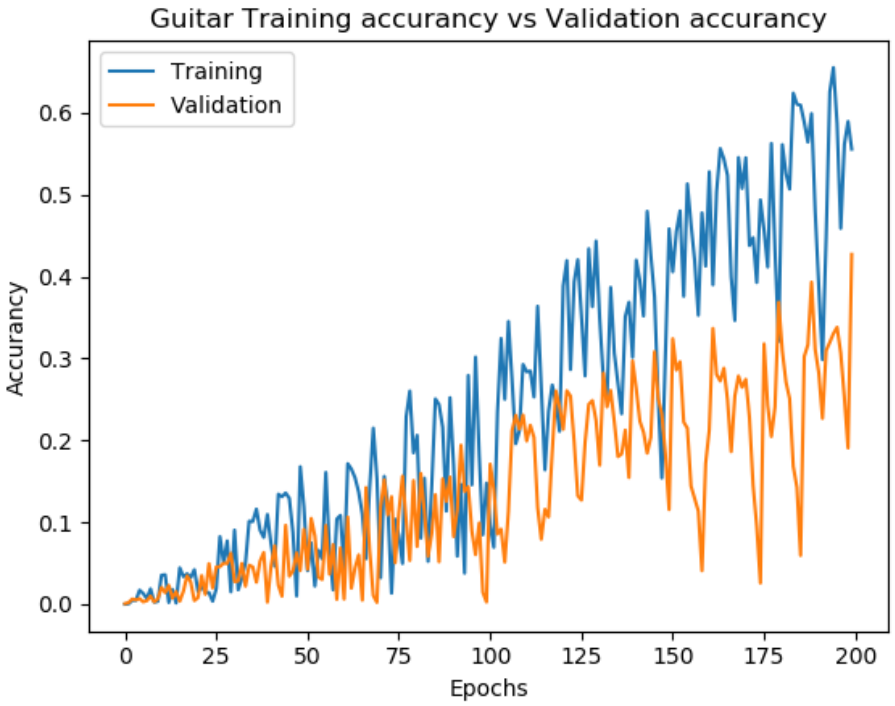

Figura 5.6: Gráfica de aprendizaje para guitarra en arquitectura 2

En la Tabla 5.6 se observa que para la fase de entrenamiento se llegó a un valor final de 55.58 % y en la fase de validación a un 42.73 %, obteniendo valores máximos en algunas épocas de hasta 65.55 % y 42.73 % respectivamente.

El desempeño de esta red para guitarra es un poco menor al de la arquitectura 1, a pesar de que el optimizador utiliza un algoritmo de momento para salir de mínimos locales.

Fase	Valor inicial	Valor final	Valor mínimo	Valor máximo
Entrenamiento	0.0000	0.5558	0.0000	0.6555
Validación	0.0007	0.4273	0.0007	0.4273

Tabla 5.6: Valores de aprendizaje para guitarra en arquitectura 2.

5.1.2.2. Bajo

En la Figura 5.7 se muestra como fue disminuyendo el error usando esta arquitectura para bajo. El error desciende de una forma más lineal que en el caso de la guitarra.

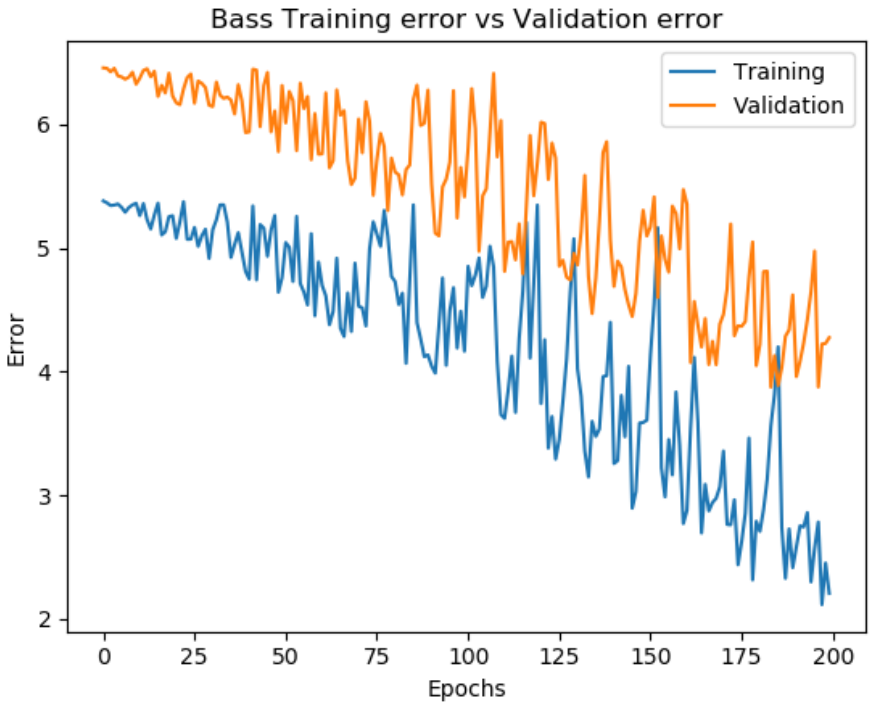

Figura 5.7: Gráfica de error para bajo en arquitectura 2

En la Tabla 5.7 se observa que el error disminuyo a niveles de 2.2061 y 4.2744 en las fases de entrenamiento y validación respectivamente, sin embargo se tuvieron valores mínimos de 2.1150 y 3.8736 en épocas anteriores.

Fase	Valor inicial	Valor final	Valor mínimo	Valor máximo
Entrenamiento	5.3796	2.2061	2.1150	5.3796
Validación	6.4556	4.2744	3.8736	6.4556

Tabla 5.7: Valores de error para bajo en arquitectura 2.

En la Figura 5.8 se muestra como fue aprendiendo la red con esta arquitectura para bajo. La fase de validación no tuvo resultados tan buenos como la fase de entrenamiento.

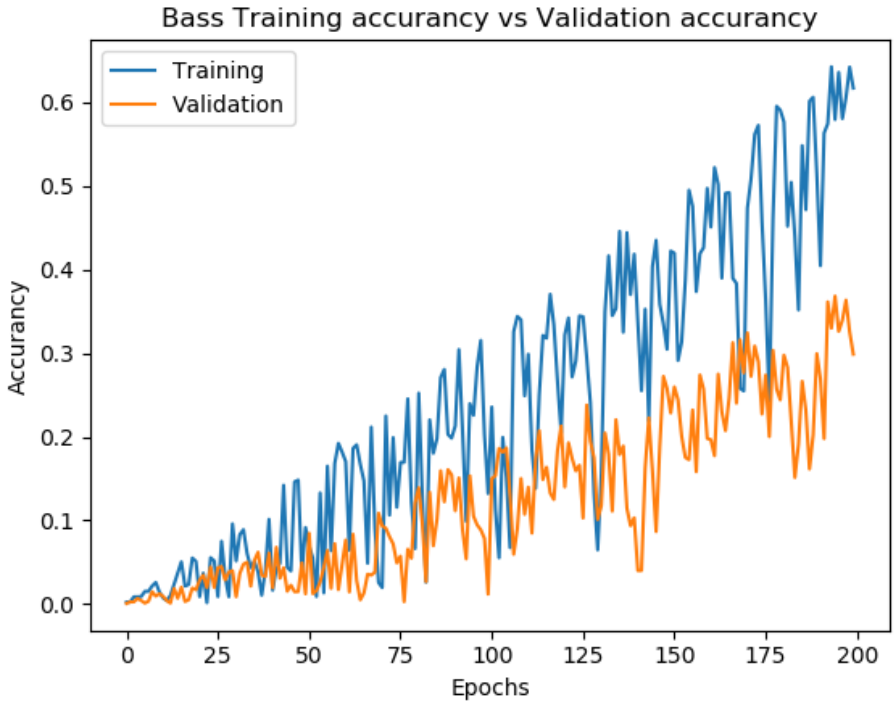

Figura 5.8: Gráfica de aprendizaje para bajo en arquitectura 2

En la Tabla 5.8 se observa que para la fase de entrenamiento se llegó a un valor final de 61.70 % y en la fase de validación a un 29.88 %, obteniendo valores máximos en algunas épocas de hasta 64.26 % y 36.83 % respectivamente.

El rendimiento en la fase de validación es de más de un 50 % menor que en la fase de entrenamiento. Esto demuestra que la red no está aprendiendo a la velocidad deseada.

La guitarra obtuvo mejores resultados de aprendizaje que el bajo en la fase de validación siendo aproximadamente un 15 % mayor.

Fase	Valor inicial	Valor final	Valor mínimo	Valor máximo
Entrenamiento	0.0024	0.6170	0.0017	0.6426
Validación	0.0008	0.2988	0.0008	0.3683

Tabla 5.8: Valores de aprendizaje para bajo en arquitectura 2.

5.1.3. Arquitectura 3

En esta arquitectura se quitaron las capas de Dropout, por lo tanto la red es menos profunda. Al igual que en la arquitectura 1 se está usando un optimizador RMSProp.

5.1.3.1. Guitarra

En la Figura 5.9 se muestra como fue cambiando el error usando esta arquitectura para guitarra. Se puede observar que el error va creciendo conforme las épocas pasan.

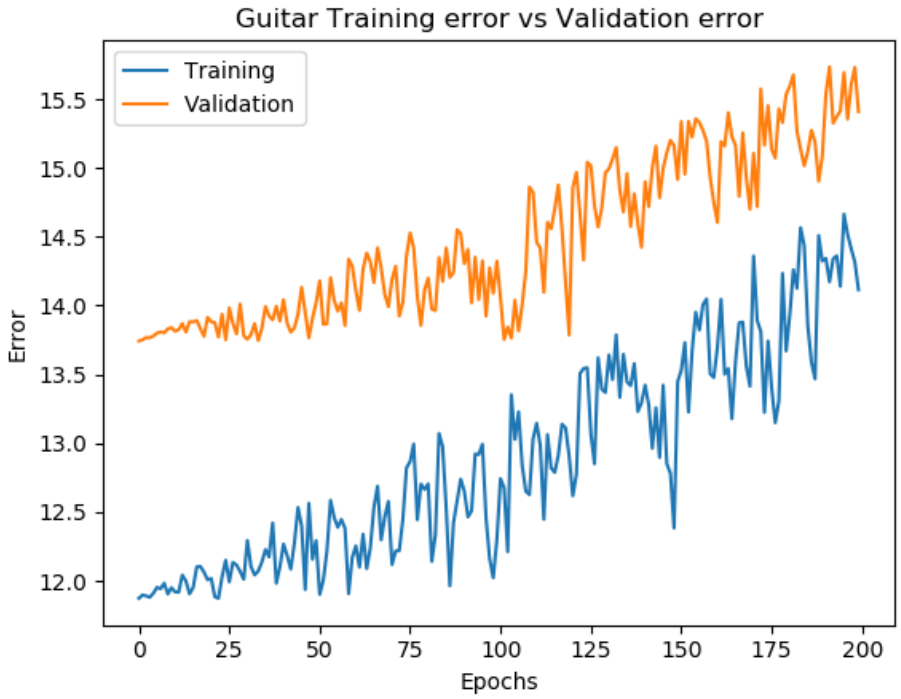

Figura 5.9: Gráfica de error para guitarra en arquitectura 3

En la Tabla 5.9 se puede observar que la tendencia del error es a la alza, llegando a valores de 14.1156 en la fase de entrenamiento y de 15.4095 en la fase de validación.

Fase	Valor inicial	Valor final	Valor mínimo	Valor máximo
Entrenamiento	11.8742	14.1156	11.8739	14.6645
Validación	13.7430	15.4095	13.7430	15.7345

Tabla 5.9: Valores de error para guitarra en arquitectura 3.

En la Figura 5.10 se muestra como fue aprendiendo la red con esta arquitectura para guitarra. A pesar de mostrar un comportamiento a la alza, los valores finales de esta gráfica no son tan alentadores.

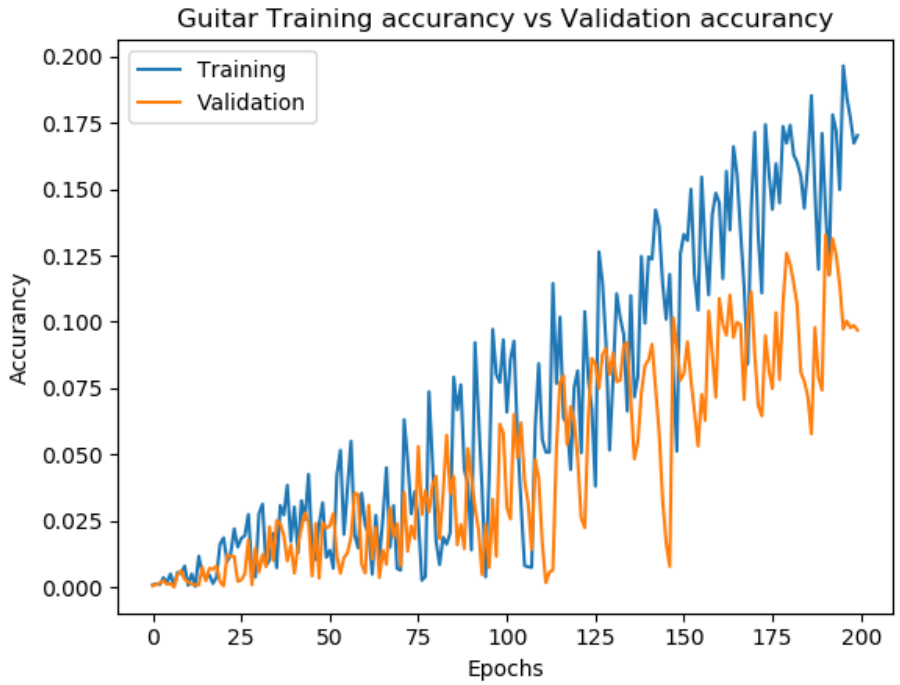

Figura 5.10: Gráfica de aprendizaje para guitarra en arquitectura 3

En la Tabla 5.10 se observa que para la fase de entrenamiento se llegó a un valor final de 17.02 % y en la fase de validación a un 9.69 %, obteniendo máximos de hasta 19.64 % y 13.28 % respectivamente.

Estos valores son bastante bajos y no garantizan que con el transcurso de las épocas esta arquitectura pueda seguir mejorando.

Fase	Valor inicial	Valor final	Valor mínimo	Valor máximo
Entrenamiento	0.0010	0.1702	0.0004	0.1964
Validación	0.0006	0.0969	0.0002	0.1328

Tabla 5.10: Valores de aprendizaje para guitarra en arquitectura 3.

5.1.3.2. Bajo

En la Figura 5.11 se muestra como fue cambiando el error usando esta arquitectura para bajo. Se puede observar un crecimiento del error conforme las épocas pasan.

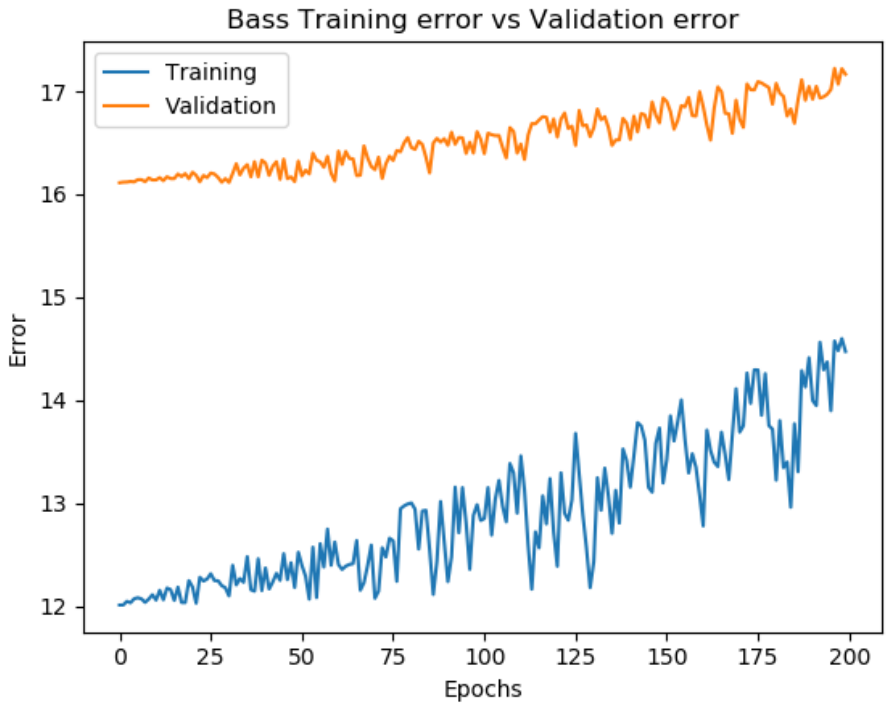

Figura 5.11: Gráfica de error para bajo en arquitectura 3

En la Tabla 5.11 se puede observar que la tendencia del error es a la alza, llegando a valores de 14.4731 en la fase de entrenamiento y de 17.1656 en la fase de validación.

Fase	Valor inicial	Valor final	Valor mínimo	Valor máximo
Entrenamiento	12.0105	14.4731	12.0105	14.5983
Validación	16.1120	17.1656	16.1120	17.2226

Tabla 5.11: Valores de error para bajo en arquitectura 3.

En la Figura 5.12 se muestra como fue aprendiendo la red con esta arquitectura para bajo. Esta gráfica muestra que el bajo va aprendiendo pero la tasa de aprendizaje es bastante baja.

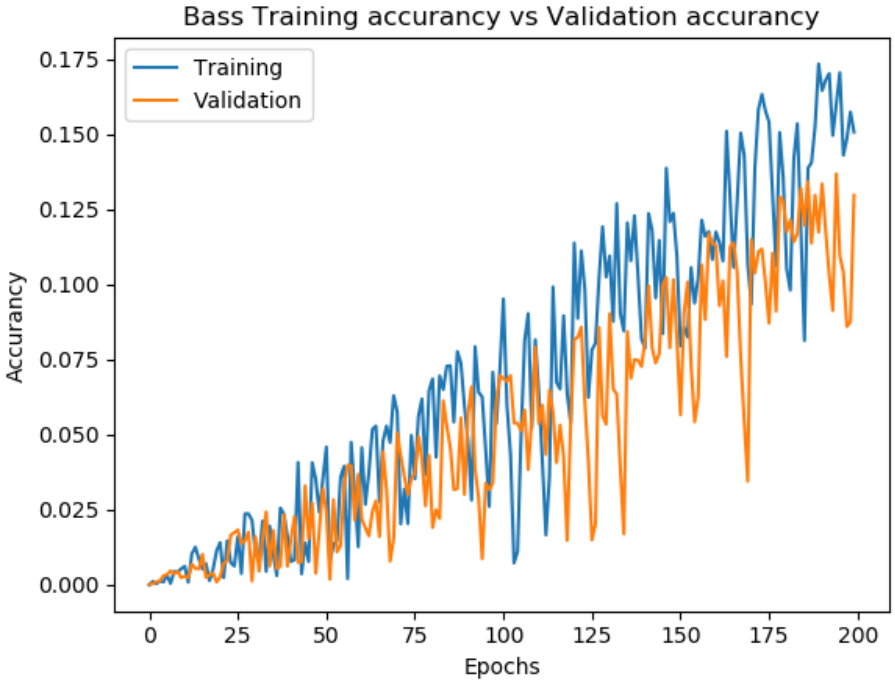

Figura 5.12: Gráfica de aprendizaje para bajo en arquitectura 3

En la Tabla 5.12 se observa que para la fase de entrenamiento se llegó a un valor final de 15.07 % y en la fase de validación a un 12.97 %, obteniendo máximos de hasta 17.34 % y 13.68 % respectivamente.

El rendimiento de esta red para bajo es menor al rendimiento para guitarra, sin embargo la tasa de aprendizaje es bastante parecida.

Es interesante ver que de la época 75 a la 135 la tasa de aprendizaje se mantiene oscilando entre los mismos valores, alcanzando por algunas épocas valores mayores.

Los valores obtenidos por esta red no son alentadores por lo que no podemos asegurarnos de que si entrenamos por más épocas esta red, sea capaz de alcanzar valores superiores al 70 %.

Fase	Valor inicial	Valor final	Valor mínimo	Valor máximo
Entrenamiento	0.0000	0.1507	0.0000	0.1734
Validación	0.0002	0.1297	0.0002	0.1368

Tabla 5.12: Valores de aprendizaje para bajo en arquitectura 3.

5.1.4. Comparativa de las arquitecturas

A continuación se presenta una comparativa de las diferentes arquitecturas, tomando en cuenta los valores descritos anteriormente.

5.1.4.1. Guitarra

En la tabla 5.13 vemos que en la fase de entrenamiento la arquitectura que mejor redujo el error fue la arquitectura 1 con un valor final de 1.3161, la arquitectura 2 dio un buen desempeño reduciendo el error hasta 1.8103 y finalmente, la arquitectura 3 presento el peor desempeño disminuyendo este valor.

Arquitectura	Valor inicial	Valor final	Valor mínimo	Valor máximo
1	6.6410	1.3161	1.3055	6.6410
2	6.1054	1.8103	1.4658	6.1138
3	11.8742	14.1156	11.8739	14.6645

Tabla 5.13: Comparativa del error en la fase de entrenamiento para guitarra

En la tabla 5.14 vemos que en la fase de validación la arquitectura que mejor redujo el error fue la arquitectura 1 con un valor final de 4.1383, la arquitectura 2 dio un buen desempeño reduciendo el error hasta 4.4766 y finalmente, la arquitectura 3 presento el peor desempeño disminuyendo este valor.

Arquitectura	Valor inicial	Valor final	Valor mínimo	Valor máximo
1	6.0144	4.1383	3.0009	6.0167
2	6.2114	4.4766	4.3697	6.2126
3	13.7430	15.4095	13.7430	15.7345

Tabla 5.14: Comparativa del error en la fase de validación de entrenamiento para guitarra

En la tabla 5.15 vemos que en la fase de entrenamiento la arquitectura que mejor aprendió fue la arquitectura 1 con un valor final de 56.66 %, seguido muy de cerca por la arquitectura 2 con un 55.58 % y finalmente, la arquitectura 3 presento el peor desempeño en este concepto.

Arquitectura	Valor inicial	Valor final	Valor mínimo	Valor máximo
1	0.0001	0.5666	0.0001	0.7226
2	0.0000	0.5558	0.0000	0.6555
3	0.0010	0.1702	0.0004	0.1964

Tabla 5.15: Comparativa del aprendizaje en la fase de entrenamiento para guitarra

En la tabla 5.16 vemos que en la fase de validación la arquitectura que mejor aprendió fue la arquitectura 1 con un valor final de 58.18 %, en segundo lugar tenemos a la arquitectura 2 con un 42.73 % y finalmente, la arquitectura 3 presento el peor desempeño en este concepto.

Arquitectura	Valor inicial	Valor final	Valor mínimo	Valor máximo
1	0.0001	0.5818	0.0001	0.5818
2	0.0007	0.4273	0.0007	0.4273
3	0.0006	0.0969	0.0002	0.1328

Tabla 5.16: Comparativa del aprendizaje en la fase de validación de entrenamiento para guitarra

5.1.4.2. Bajo

En la tabla 5.17 vemos que en la fase de entrenamiento la arquitectura que mejor redujo el error fue la arquitectura 2 con un valor final de 2.2061, la arquitectura 1 dio un buen desempeño reduciendo el error hasta 2.3281 y finalmente, la arquitectura 3 presento el peor desempeño disminuyendo este valor.

Arquitectura	Valor inicial	Valor final	Valor mínimo	Valor máximo
1	5.7464	2.3281	2.2614	5.7531
2	5.3796	2.2061	2.1150	5.3796
3	12.0105	14.4731	12.0105	14.5983

Tabla 5.17: Comparativa del error en la fase de entrenamiento para bajo

En la tabla 5.18 vemos que en la fase de validación la arquitectura que mejor redujo el error fue la arquitectura 1 con un valor final de 2.8547, la arquitectura 2

no redujo el error como era de esperarse quedando con un valor final de 4.2744 y finalmente, la arquitectura 3 presento el peor desempeño disminuyendo este valor.

Arquitectura	Valor inicial	Valor final	Valor mínimo	Valor máximo
1	6.0615	2.8547	2.8547	6.0620
2	6.4556	4.2744	3.8736	6.4556
3	16.112	17.1656	16.112	17.2226

Tabla 5.18: Comparativa del error en la fase de validación de entrenamiento para bajo

En la tabla 5.19 vemos que en la fase de entrenamiento la arquitectura que mejor aprendió fue la arquitectura 1 con un valor final de 74.08 %, seguido por la arquitectura 2 con un 61.70 % y finalmente, la arquitectura 3 presento el peor desempeño en este concepto.

Arquitectura	Valor inicial	Valor final	Valor mínimo	Valor máximo
1	0.0034	0.7408	0.0034	0.7408
2	0.0024	0.6170	0.0017	0.6426
3	0.0000	0.1507	0.0000	0.1734

Tabla 5.19: Comparativa del aprendizaje en la fase de entrenamiento para bajo

En la tabla 5.20 vemos que en la fase de validación la arquitectura que mejor aprendió fue la arquitectura 1 con un valor final de 52.28 %, en segundo lugar tenemos a la arquitectura 2 con un 29.88 % y finalmente, la arquitectura 3 presento el peor desempeño en este concepto.

Arquitectura	Valor inicial	Valor final	Valor mínimo	Valor máximo
1	0.0014	0.5228	0.0014	0.6170
2	0.0008	0.2988	0.0008	0.3683
3	0.0002	0.1297	0.0002	0.1368

Tabla 5.20: Comparativa del aprendizaje en la fase de validación de entrenamiento para bajo

5.2. Etapa de generación

En esta etapa se evaluara la salida de las redes desde un concepto musical. Se verá como están interactuando las notas en una salida de nuestra red.

Se tomaran muestras de pocos compases de las salidas de nuestras redes, esto para ser más objetivos con la interpretación de los resultados.

Cada red fue alimentada con las notas de una canción base, y con esta entrada más lo que la red pudo aprender en el proceso de entrenamiento generara una salida.

5.2.1. Arquitectura 1

Para la interpretación de resultados, en esta red se usó una canción base con tonalidad de Mi Mayor, y se espera que la salida posea esta misma tonalidad.

En la figura 5.13 se muestra la entrada de la red para la parte de guitarra.

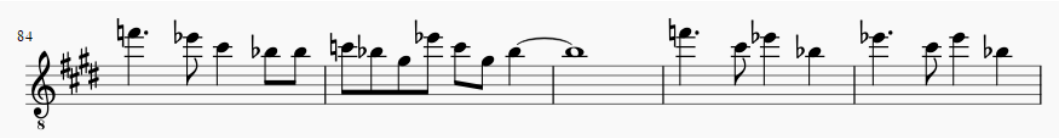

Figura 5.13: Entrada de guitarra para la arquitectura 1

En la figura 5.14 se muestra la entrada de la red para la parte de bajo.

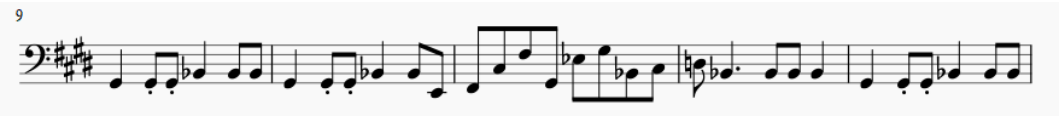

Figura 5.14: Entrada de bajo para la arquitectura 1

Una vez introducidas estas 2 entradas, la salida correspondiente que obtenemos se puede ver en la figura 5.15

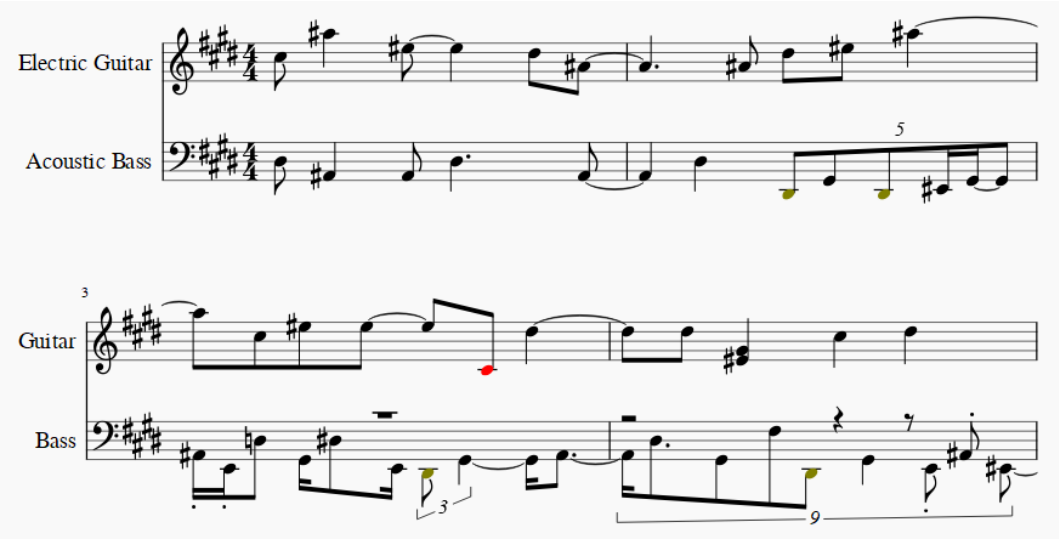

Figura 5.15: Salida de la arquitectura 1

Si analizamos la salida obtenida nos daremos cuenta que efectivamente esta nueva pieza musical conserva su tonalidad de Mi Mayor (esto se puede observar por su armadura). Se puede observar que la salida posee mayormente notas semicorcheas y negras, lo cual se asemeja mucho a la entrada original.

En el primer compas vemos que cuando la entrada en la guitarra es un Fa, la salida de nuestra red nos está entregando un Do#, pero si vemos en el 3er compas la primera nota de entrada vuelve a ser un Fa, pero en este caso obtenemos como salida un La, esto es un dato interesante ya que la red no se está basando en únicamente una nota de entrada, sino que toma en cuenta toda la secuencia.

En el caso del bajo en nuestro primer compas tenemos como primera entrada un Sol# y de salida obtenemos un Re#, en el segundo compas la red vuelve a recibir como entrada un Sol#, pero en este caso nos arroja un La.

5.2.2. Arquitectura 2

Para la interpretación de resultados, en esta red se uso una canción base con tonalidad de Do Mayor, y se espera que la salida posea esta misma tonalidad.

En la figura 5.16 se muestra la entrada de la red para la parte de guitarra.

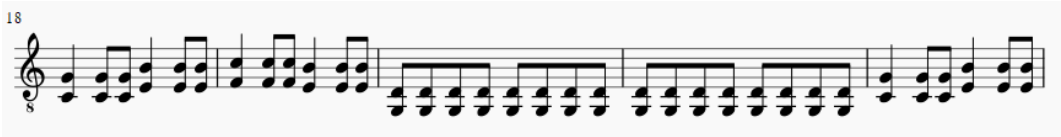

Figura 5.16: Entrada de guitarra para la arquitectura 2

En la figura 5.17 se muestra la entrada de la red para la parte de bajo.

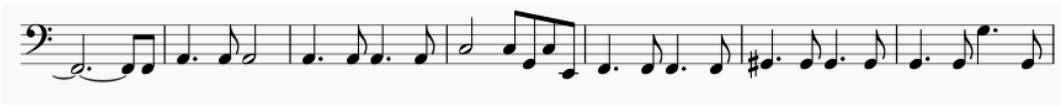

Figura 5.17: Entrada de bajo para la arquitectura 2

Una vez introducidas estas 2 entradas, la salida correspondiente que obtenemos se puede ver en la figura 5.18

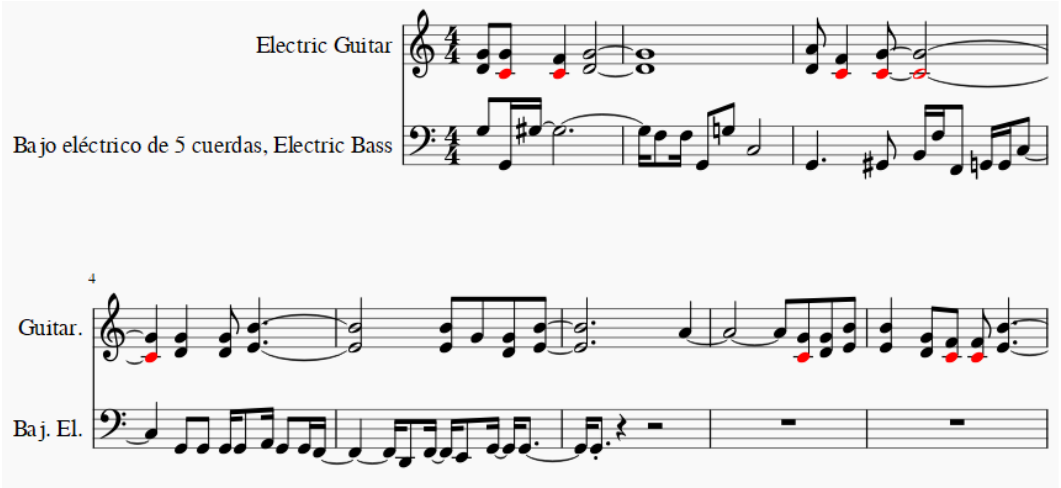

Figura 5.18: Salida de la arquitectura 2

La nueva pieza musical posee la tonalidad de Do Mayor, tal como las secuencias de entradas, lo que nos indica que esta red está trabajando correctamente.

La salida que estamos obteniendo con esta red usa principalmente notas de corcheas y negras, lo cual corresponde muy bien con las notas de entrada que está recibiendo la red.

Si realizamos un análisis de las notas, tenemos, que en la parte de guitarra en nuestro primer compas tenemos un acorde de poder formado por Do y Sol, y a la salida

obtenemos también un acorde de poder pero conformado por Re y Sol, la siguiente nota de entrada es nuevamente el acorde de Do y Sol, y en este caso obtenemos el mismo acorde de salida.

En cuanto a la salida de bajo se puede observar que posee semicorcheas aparte de corcheas y negras, esto puede ser debido a las otras melodías que aprendió la red.

La primer nota de entrada que tenemos en el bajo es Fa y nos genera un Mi, después en el quinto compas volvemos a introducir un Fa y la salida obtenida resulta ser otro Fa.

En base a estos argumentos podemos verificar que esta red está cumpliendo su cometido de aprender secuencias y generar salidas que tengan sentido con las secuencias de entrada.

5.2.3. Arquitectura 3

Para la interpretación de resultados, en esta red se usó una canción base con tonalidad de Sol Mayor, y se espera que la salida posea esta misma tonalidad.

En la figura 5.19 se muestra la entrada de la red para la parte de guitarra.

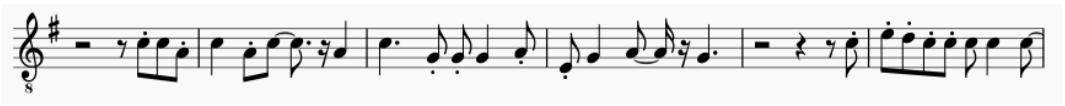

Figura 5.19: Entrada de guitarra para la arquitectura 3

En la figura 5.20 se muestra la entrada de la red para la parte de bajo.

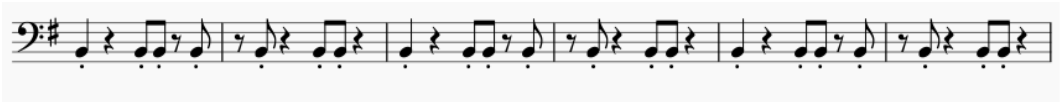

Figura 5.20: Entrada de bajo para la arquitectura 3

Una vez introducidas estas 2 entradas, la salida correspondiente que obtenemos se puede ver en la figura 5.21

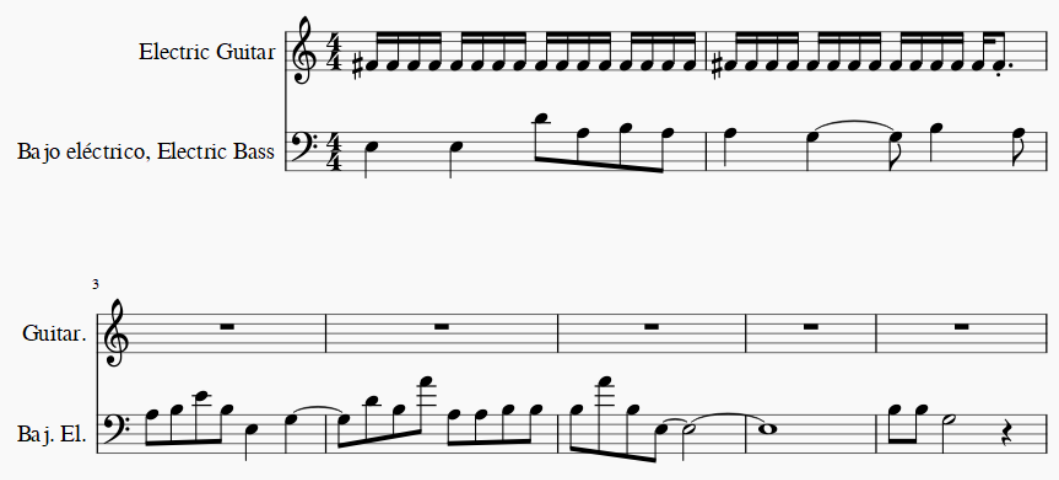

Figura 5.21: Salida de la arquitectura 3

La nueva pieza musical generada parece tener la misma tonalidad de Sol Mayor, ya que posee una alteración en Fa#, sin embargo en el bajo la tonalidad predominante es Do Mayor.

La salida en el caso de la guitarra son puras semicorcheas, mientras que en el bajo son de corcheas y negras. Cabe mencionar que las notas de entrada se componen principalmente de corcheas y negras.

Se observa que la salida de la guitarra es bastante pobre, ya que únicamente está generando notas de Fa# sin importar la entrada.

En la parte del bajo se observa que tiene mayor ritmo que la guitarra, sin embargo si analizamos la entrada, son puras secuencias de Si, y a la salida obtenemos notas que parecieran sacadas al azar.

5.2.4. Comparativa de las arquitecturas

En base a lo que se describió de los resultados de cada una de las arquitecturas, musicalmente hablando las arquitecturas 1 y 2 presentaron resultados alentadores, no es así el caso para la arquitectura 3.

Mientras que en la arquitectura 1 y 2 se puede observar valores de notas y ritmos similares a las entradas, la arquitectura 3 muestra puros patrones aleatorios.

5.3. Etapa de validación

Durante esta etapa se evaluaron 100 canciones creadas usando las diferentes arquitecturas y aplicándoles el algoritmo de Krumhansl-Schmuckler para determinar la tonalidad final de las notas de cada instrumento.

Cada composición es hecha a partir de una canción base, la cual posee una tonalidad ya establecida y la red debe de ser capaz de crear una nueva composición tomando como base las notas de esta canción, la nueva pieza debería tener la misma tonalidad que la canción base, esto nos indica que nuestra red es capaz de identificar tonalidades y de crear nuevas cosas en base a estas.

La salida de cada instrumento es generado de manera independiente usando el Track correspondiente de la canción base.

5.3.1. Arquitectura 1

Para la arquitectura 1 se obtuvieron los siguientes resultados:

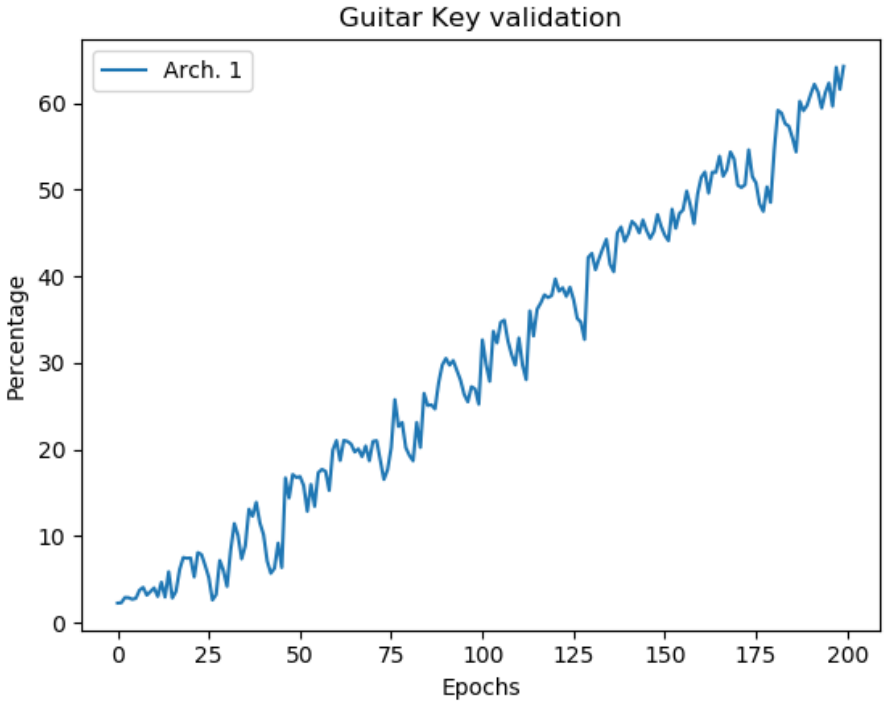

Figura 5.22: Gráfica de validación para guitarra en arquitectura 1

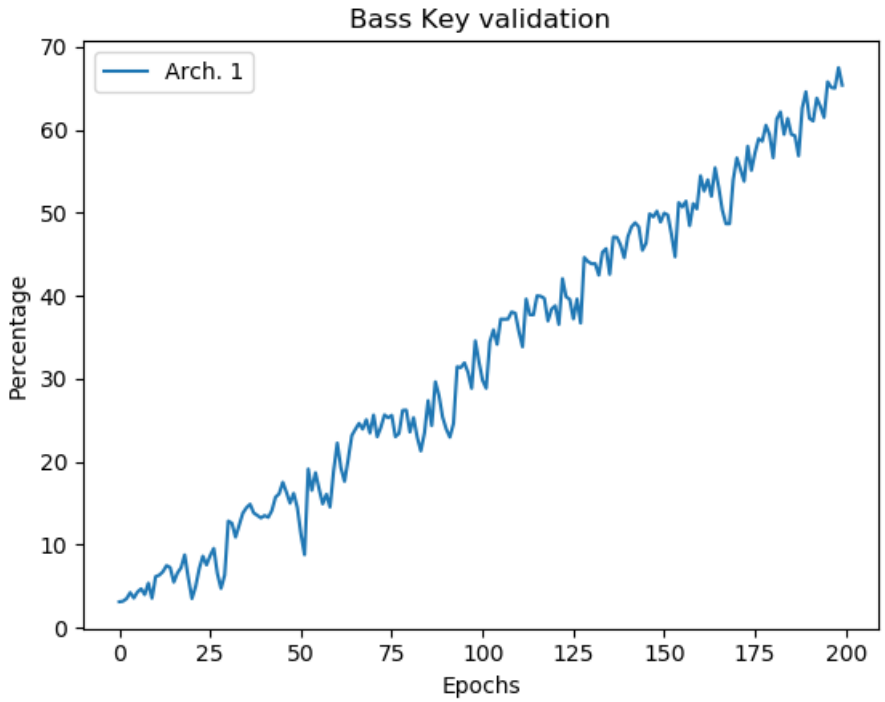

Figura 5.23: Gráfica de validación para bajo en arquitectura 1

En las figuras 5.22 y 5.23 se puede observar que conforme van pasando las épocas, el número de canciones creadas que poseen la misma tonalidad que la canción base va en aumento.

En la tabla 5.21 vemos que los valores alcanzados por esta arquitectura son de 64.26 % para guitarra y de 65.38 % para bajo, teniendo unos máximos de 64.26 % y 67.50 % respectivamente.

Instrumento	Valor inicial	Valor final	Valor mínimo	Valor máximo
Guitarra	2.2510	64.2632	2.2510	64.2632
Bajo	3.1204	65.3834	3.1204	67.5041

Tabla 5.21: Valores de validación para la arquitectura 1.

5.3.2. Arquitectura 2

Para la arquitectura 2 se obtuvieron los siguientes resultados:

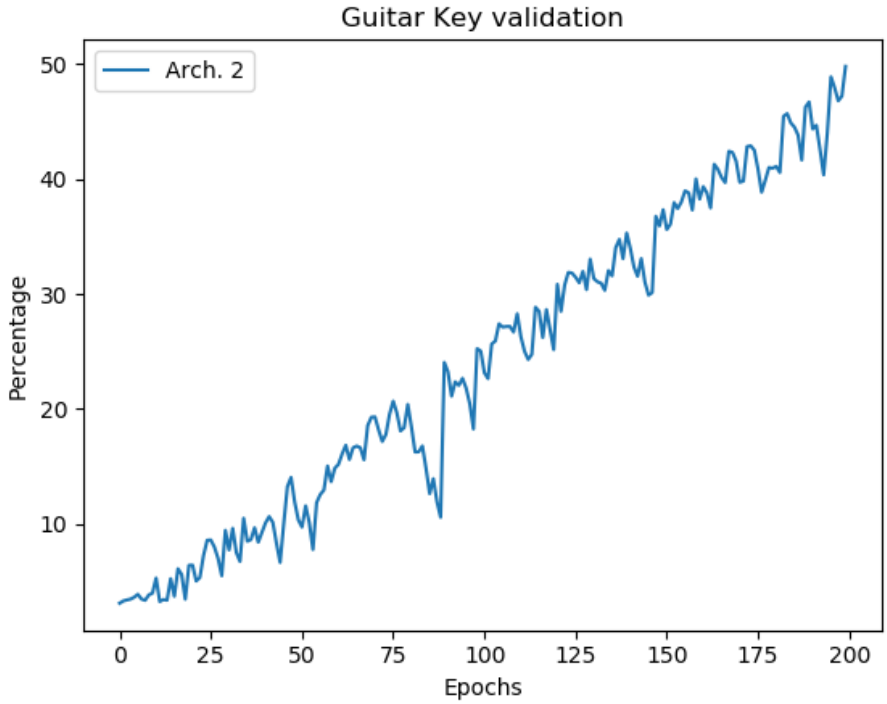

Figura 5.24: Gráfica de validación para guitarra en arquitectura 2

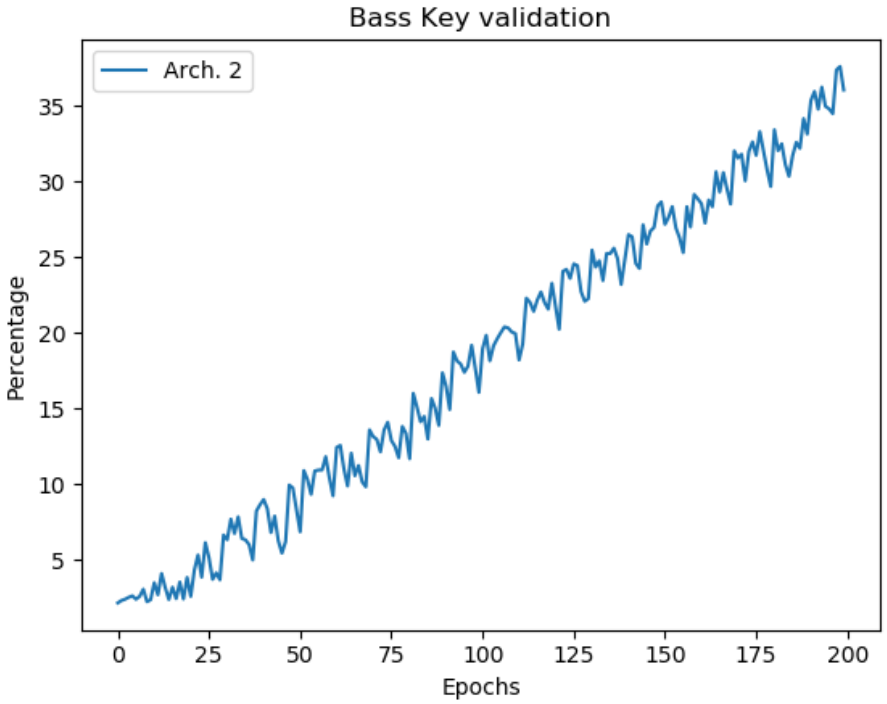

Figura 5.25: Gráfica de validación para bajo en arquitectura 2

En las figuras 5.24 y 5.25 se puede observar que conforme van pasando las épocas, el número de canciones creadas que poseen la misma tonalidad que la canción base va en aumento, sin embargo el crecimiento en las canciones de bajo es mucho más lento en comparación a la guitarra.

En la tabla 5.22 vemos que los valores alcanzados por esta arquitectura son de 49.81 % para guitarra y de 36.07 % para bajo, teniendo unos máximos de 49.81 % y 37.62 % respectivamente.

Instrumento	Valor inicial	Valor final	Valor mínimo	Valor máximo
Guitarra	3.1008	49.8122	3.1008	49.8122
Bajo	2.1270	36.0719	2.1270	37.6297

Tabla 5.22: Valores de validación para la arquitectura 2.

5.3.3. Arquitectura 3

Para la arquitectura 3 se obtuvieron los siguientes resultados:

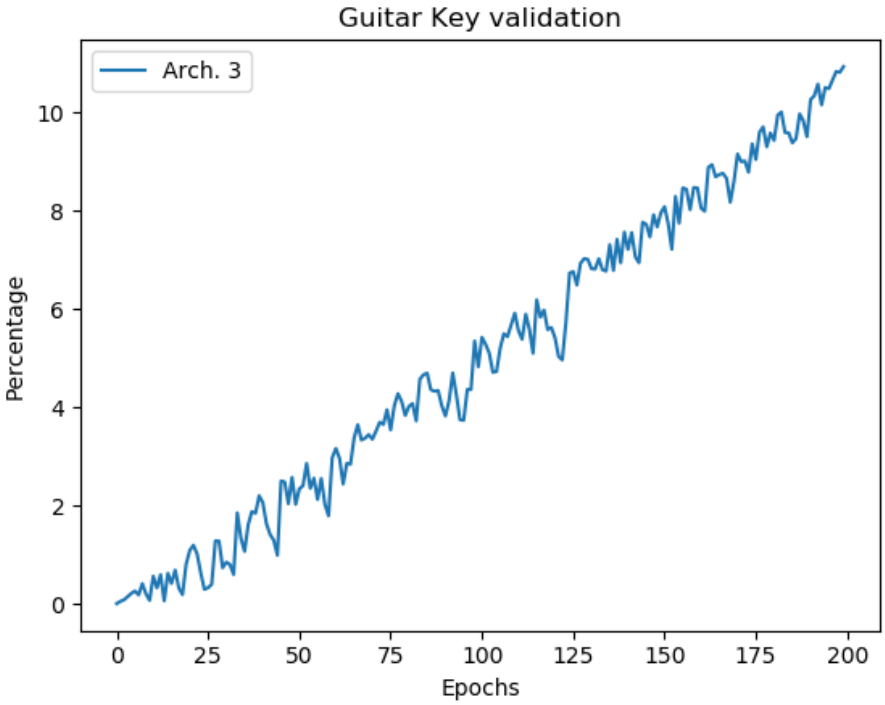

Figura 5.26: Gráfica de validación para guitarra en arquitectura 3

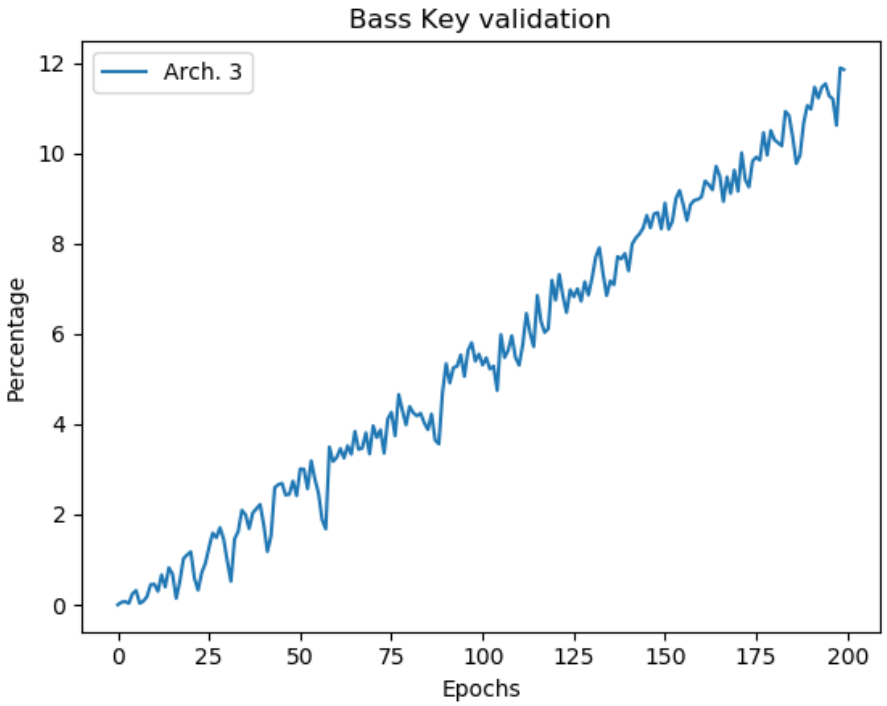

Figura 5.27: Gráfica de validación para bajo en arquitectura 3

En las figuras 5.26 y 5.27 se puede observar que conforme van pasando las épocas, el número de canciones creadas que poseen la misma tonalidad que la canción base va en aumento, sin embargo el porcentaje final es bastante bajo después de 200 épocas.

En la tabla 5.23 vemos que los valores alcanzados por esta arquitectura son de 10.94 % para guitarra y de 11.85 % para bajo, teniendo unos máximos de 10.94 % y 11.88 % respectivamente.

Instrumento	Valor inicial	Valor final	Valor mínimo	Valor máximo
Guitarra	0.0021	10.9422	0.0021	10.9422
Bajo	0.0022	11.8519	0.0022	11.8865

Tabla 5.23: Valores de validación para la arquitectura 3.

5.3.4. Comparativa de las arquitecturas

Podemos realizar una comparativa de estas arquitecturas si se grafican juntas.

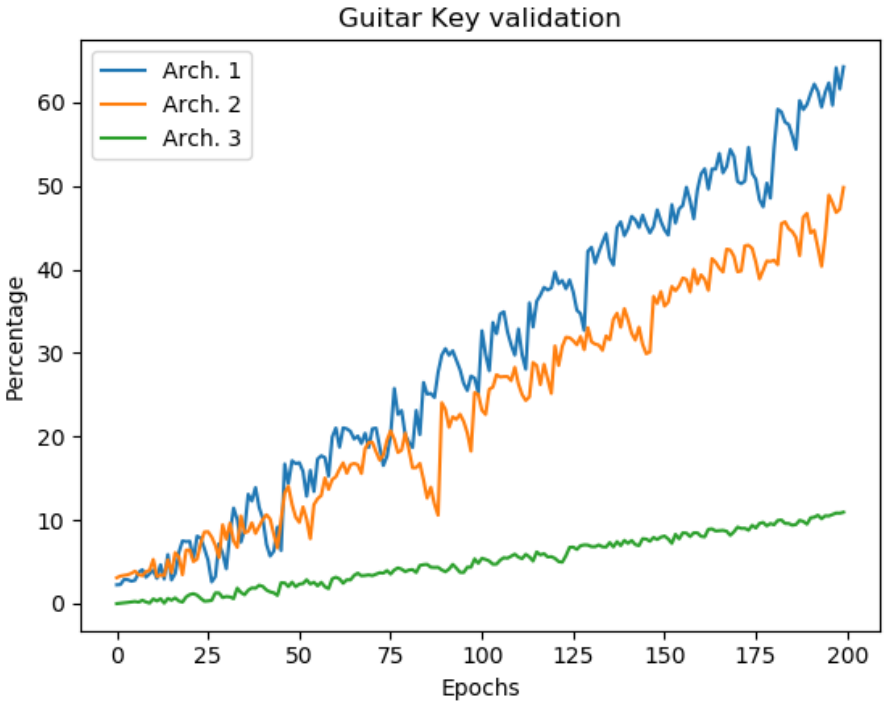

Figura 5.28: Gráfica comparativa de arquitecturas para guitarra

En la figura 5.28 observamos que la arquitectura que tuvo mejores resultados fue la arquitectura 1, seguido de la arquitectura 2. El porcentaje de la arquitectura 3 es bastante bajo después de todas estas épocas.

En la tabla 5.24 se observa que los valores alcanzados por cada arquitectura fueron de 64.26 %, 49.81 % y 10.94 % respectivamente.

Arquitectura	Valor inicial	Valor final	Valor mínimo	Valor máximo
1	2.2510	64.2632	2.2510	64.2632
2	3.1008	49.8122	3.1008	49.8122
3	0.0021	10.9422	0.0021	10.9422

Tabla 5.24: Comparativa de arquitecturas en la fase de validación para guitarra.

En la figura 5.29 se muestra el comportamiento que tuvieron las arquitecturas para bajo, siendo de nuevo la mejor la arquitectura 1.

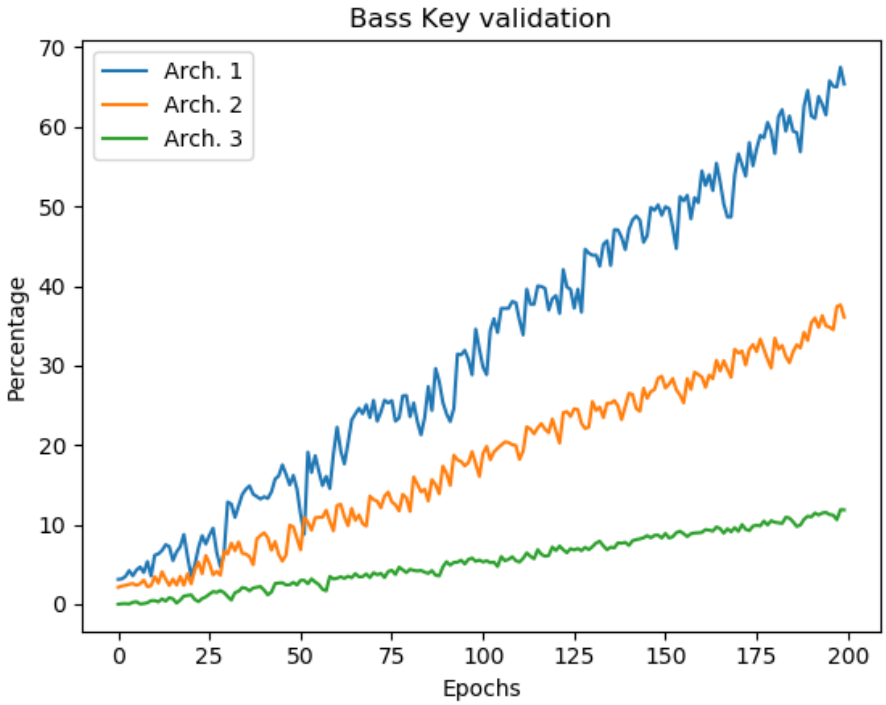

Figura 5.29: Gráfica comparativa de arquitecturas para bajo

En la tabla 5.25 se observa que los valores alcanzados por cada arquitectura fueron de 65.38 %, 36.07 % y 11.85 % respectivamente.

Arquitectura	Valor inicial	Valor final	Valor mínimo	Valor máximo
1	3.1204	65.3834	3.1204	67.5041
2	2.1270	36.0719	2.1270	37.6297
3	0.0022	11.8519	0.0022	11.8865

Tabla 5.25: Comparativa de arquitecturas en la fase de validación para bajo.

Capítulo 6
Conclusiones

Las diferentes arquitecturas presentadas en este documento fueron capaces de aprender y generar música a partir de secuencias de notas de entrada. A pesar de que algunas arquitecturas no tuvieron el desempeño esperado, los resultados finales fueron bastante alentadores.

Estas redes fueron capaces de generar tanto melodías como armonías musicales en guitarra y melodías básicas de bajo.

Las secuencias de salida generadas por las redes poseen los 2 elementos básicos de la música: sonidos y silencios. Sin embargo lo interesante de estas salidas es ver cómo están interactuando estos elementos para finalmente crear una nueva pieza musical.

Al parecer las secuencias de guitarra fueron más sencillas de aprender que las de bajo, esto es un dato bastante curioso, puesto que en bajo no se tenían tantos cambios de tonos como en la guitarra.

Después de todas las pruebas realizadas a las arquitecturas, la que tuvo el mejor desempeño fue sin duda la arquitectura 1, la cual cuenta con capaz intermedias de Dropout así como también un optimizador RMSProp. Sin embargo la arquitectura 2 también presento resultados bastante alentadores.

Es interesante analizar que aunque se use un optimizador con momento como el Nadam, este no genera un resultado sobresaliente en redes LSTM, al parecer entre más simple sea el optimizador mejores resultados se obtienen.

Las capas de Dropout fueron vitales para el correcto aprendizaje de estas redes, ya que sin ellas, el error iba aumentando en lugar de disminuir al transcurrir las épocas. Este comportamiento lo vemos más claramente en la arquitectura 3.

Dentro de la búsqueda por encontrar un algoritmo computacional que nos permitiera realizar una validación de la salida, se encontró con el algoritmo de Krumhansl-Schmuckler, que a pesar, de ser bastante sencillo, nos genera muy buenos valores de

validación, lo que nos permitió dar certeza de que las piezas generadas por estas redes, estaban cumpliendo con lo establecido.

A pesar de que las redes se entrenaron por separado para guitarra y bajo, la salida que se obtiene al conjuntar la salida de ambos instrumentos se escucha bastante bien.

El tiempo de procesamiento es algo a tomar en cuenta cuando se trabajan con muchos datos y este tipo de redes, ya que computacionalmente hablando el procesamiento de estas redes es bastante costoso, y entre mas capas se estén usando en las redes, más tiempo tarda por cada época de entrenamiento.

En este trabajo únicamente se realizaron 200 épocas de cada instrumento por cada una de las arquitecturas, ya que el tiempo promedio de procesamiento era de alrededor de 3 semanas por instrumento, lo que en conjunto de todas las pruebas realizadas se tomó cerca de 5 meses.

6.1. Trabajos futuros

Dentro de los trabajos futuros de esta investigación seria bueno experimentar con otro tipo de arquitecturas usando capas LSTM, para ver si se logran mejores resultados a los obtenidos en este trabajo.

La infinidad de posibilidades de nuevas arquitecturas solamente se limita a la imaginación del ser humano, por lo tanto no dudo que en un futuro se creen arquitecturas más complejas para el procesamiento de secuencias. Posiblemente estas arquitecturas sean capaces de aprender y obtener buenos resultados en menor cantidad de épocas.

Conforme el tiempo pasa, el hardware también sufre modificaciones haciendo a los equipos más potentes para procesar grandes cantidades de datos, es posible que en un futuro no muy lejano se puedan procesar estas redes en cuestión de horas, en lugar de días o semanas, eso también nos abriría la posibilidad de introducir una mayor cantidad de datos o de inclusive usar arquitecturas más robustas, sin importarnos tanto el tiempo de procesamiento.

Sería bueno en un trabajo futuro analizar estas mismas arquitecturas usando un algoritmo de validación de armonías musicales, para ver todo el campo armónico y dar una mayor certeza de que se estas redes están generando música de calidad. Este trabajo únicamente se centró en entrenar a las redes para que fueran capaces de aprender tonalidades y generar música a partir de ellas.

Es posible extrapolar este proyecto a otros instrumentos de cuerda o de viento,

tales como violín, saxofón, trompeta, entre otros. El método de aprendizaje sería muy similar, lo único que se tendría que hacer es que en la etapa de preprocesamiento, separar los Tracks de estos instrumentos.

La batería y los instrumentos de percusión se manejan un poco diferente a los demás instrumentos, por lo que sería interesante ver si una red es capaz de realizar ritmos bases, los cuales puedan ser integrados a nuevas canciones, así como también ver si se generan nuevos ritmos diferentes a los que la red aprendió.

Es posible entrenar estas redes con otros estilos musicales, lo único que se necesita es una base de datos bastante grande de canciones de estos géneros, pero básicamente todas las demás etapas no sufrirían cambios.

Bibliografía

[1] H. Lee, Y. Largman, P. Pham, and A. Y. Ng, "Unsupervised feature learning for audio classification using convolutional deep belief networks," in *Proceedings of the 22Nd International Conference on Neural Information Processing Systems*, ser. NIPS'09. USA: Curran Associates Inc., 2009, pp. 1096–1104. [Online]. Available: http://dl.acm.org/citation.cfm?id=2984093.2984217

[2] E. J. Humphrey, J. P. Bello, and Y. Lecun, "Feature learning and deep architectures: New directions for music informatics," *J. Intell. Inf. Syst.*, vol. 41, no. 3, pp. 461–481, Dec. 2013. [Online]. Available: http://dx.doi.org/10.1007/s10844-013-0248-5

[3] X. Wang and Y. Wang, "Improving content-based and hybrid music recommendation using deep learning," in *Proceedings of the 22Nd ACM International Conference on Multimedia*, ser. MM '14. New York, NY, USA: ACM, 2014, pp. 627–636. [Online]. Available: http://doi.acm.org/10.1145/2647868.2654940

[4] A. Huang and R. Wu, "Deep learning for music," *CoRR*, vol. abs/1606.04930, 2016. [Online]. Available: http://arxiv.org/abs/1606.04930

[5] J. Pons, O. Nieto, M. Prockup, E. M. Schmidt, A. F. Ehmann, and X. Serra, "End-to-end learning for music audio tagging at scale," *CoRR*, vol. abs/1711.02520, 2017. [Online]. Available: http://arxiv.org/abs/1711.02520

[6] J. Briot, G. Hadjeres, and F. Pachet, "Deep learning techniques for music generation - A survey," *CoRR*, vol. abs/1709.01620, 2017. [Online]. Available: http://arxiv.org/abs/1709.01620

[7] C. Raffel, "Learning-based methods for comparing sequences, with applications to audio-to-midi alignment and matching," Ph.D. dissertation, Columbia University, 2016.

[8] S. J. Puig, *Audio digital y MIDI*. Guias Monograficas Anaya Multimedia, 1997.

[9] M. H. Hassoun, *Fundamentals of Artificial Neural Networks*, 1st ed. Cambridge, MA, USA: MIT Press, 1995.

[10] K. Gurney, *An Introduction to Neural Networks*. Bristol, PA, USA: Taylor & Francis, Inc., 1997.

[11] I. Nunes, D. Hernane, R. Andrade, L. Bartocci, and S. dos Reis, *Artificial Neural Networks: A Practical Course*, Springer, Ed., 2017.

[12] N. Buduma, *Fundamentals of Deep Learning: Designing Next-Generation Machine Intelligence Algorithms*, O. Reilly, Ed., 2017.

[13] L. Deng and D. Yu, *Deep Learning: Methods and Applications*. Hanover, MA, USA: Now Publishers Inc., 2014.

[14] E. Herrera, *Teoría musical y armonía moderna*. Bosch, 1991. [Online]. Available: https://books.google.com.mx/books?id=K5Qmp0GjJIEC

[15] E. Taylor and A. B. of the Royal Schools of Music (Great Britain), *The AB Guide to Music Theory*, ser. The AB Guide to Music Theory. Associated Board of the Royal Schools of Music, 1989, no. parte 1. [Online]. Available: https://books.google.es/books?id=h7hZSAAACAAJ