Escuela Técnica Superior de Ingenieros Informáticos

Universidad Politécnica de Madrid

Redes neuronales recurrentes para la generación automática de música

Trabajo Fin de Máster

Máster Universitario en Inteligencia Artificial

AUTOR: Juan Julián Cea Morán

TUTOR: Francisco Serradilla García

2020

II

Agradecimientos

En primer lugar he de agradecer a mi familia y en especial a mis padres, Juan Julián y Alejandra así como a mi hermana Irene. Gracias a su cariño y apoyo incondicional durante toda mi vida, tanto en mis decisiones académicas como personales. También a mi abuelo Chan, por confiar siempre en los inventos de su nieto.

En segundo lugar, me gustaría agradecer a mis compañeros de piso Enrique, Pedro y Patricia, quienes me han acompañado durante parte de mi vida y en especial durante este periodo de confinamiento, haciendo de este año algo inolvidable.

En tercer lugar, agradecer su atención y esfuerzo a mi tutor Francisco, quien ha estado en todo momento dispuesto a ayudarme y aconsejarme ante cualquier problema encontrado durante el desarrollo de esta tesis.

Por último, agradecer a la Universidad Politécnica de Madrid por los servicios y recursos prestados. En especial, a Javier Bajo, por su trato cercano y sus consejos durante todo el curso.

Índice general

Índice de figuras

Resumen

Usualmente se tiende a pensar en los sistemas informáticos como elementos puramente objetivos y deterministas, capaces de generar cierta salida dada cierta entrada. Dentro del campo de la Inteligencia Artificial, se pretenden resolver problemas del mismo modo que lo haría un humano, y en muchas ocasiones eso requiere que las técnicas y herramientas usadas sean puramente objetivas. Sin embargo, existen problemáticas que no pueden ser resueltas de esta forma, si no que tal y como lo harían los humanos, necesitan de cierta creatividad. El problema de la generación musical automática, es uno de estos supuestos en los que se desea que el sistema que lo resuelva, sea capaz de plasmar cierta creatividad. Afortunadamente existen una gran variedad de algoritmos y métodos que permiten realizar esta tarea. Dentro del campo del *Deep Learning*, existe un tipo de arquitectura neuronal que destaca por encima del resto: las Redes Neuronales Recurrentes. En esta tesis se pretende entender este tipo de arquitecturas, explicando por qué son tan populares en el campo de la composición musical automática. Además, se implementan algunas topologías con el fin de ejemplificar lo aprendido y demostrar que efectivamente existen modelos capaces de aportar soluciones creativas a problemas humanos.

Abstract

There is usually a tendency to think of computer systems as purely objective and deterministic elements, capable of generating a certain output given a certain input. Within the field of Artificial Intelligence, the aim is to solve problems in the same way that a human would, and on many occasions this requires the techniques and tools used to be purely objective. However, there are problems that cannot be solved in this way, but rather just as humans would, they need a certain creativity. The problem of automatic music generation is an example in which we want the system that solves it to be able to show some creativity. Fortunately, there is a great variety of algorithms and methods that allow us to carry out this task. In the Deep Learning field, there is a type of neuronal architecture that stands out from the rest: the Recurrent Neuronal Networks. This thesis aims to understand this type of architecture, explaining why they are so popular for automatic music composition. In addition, some topologies are implemented in order to exemplify what has been learned and demonstrate that there are indeed algorithms capable of providing creative solutions to human problems.

1 Introducción

Desde hace siglos, los seres humanos se han interesado por la invención de técnicas y sistemas que permitiesen facilitar el proceso de composición musical. Con la aparición de los primeros computadores esta tendencia se incrementó, dando lugar a la creación de una gran cantidad de herramientas que permiten asistir a los compositores así como métodos algorítmicos que generan música con cierto carácter creativo.

Por otra parte, desde sus inicios Internet se ha constituido como una gran fuente de información. Concretamente, en la última década se ha producido un incremento masivo en la cantidad de datos generados sobre cualquier dominio. Este gran aumento ha propiciado el auge de una gran variedad de tecnologías dentro del área de la Inteligencia Artificial, ya que muchos de estos modelos obtienen resultados más precisos cuantos más datos se usen para su entrenamiento.

Uno de los modelos que más beneficiado se ha visto son las redes de neuronas artificiales y en concreto las técnicas de *Deep Learning*. Las técnicas de *Deep Learning* han demostrado un gran éxito en tareas como clasificación, procesamiento de imágenes o de sonido, etc. Del mismo modo destacan cuando se habla de tareas de carácter creativo, como pueden ser generación automática de texto o composición automática de música.

1.1 Objetivos

El objetivo principal de esta tesis de fin de máster consiste en investigar las técnicas de *Deep Learning* y en concreto las redes de neuronas artificiales recurrentes como herramienta para la composición automática de música.

1.1.1 Análisis del estado del arte

En primer lugar, se realizará una revisión de las principales aportaciones al dominio de la generación musical automática y de las diversas tecnologías que se han usado a lo largo de la historia. Todo esto, conducirá al estudio del campo en el que se centra esta tesis: el *Deep Learning*. Se revisarán las principales arquitecturas capaces de generar contenido musical de manera creativa, centrándose en las redes neuronales recurrentes ya que son el principal exponente en el área.

1.1.2 Implementación de arquitecturas

Una vez revisadas las propuestas que conforman el estado del arte en el campo de la composición automática, se procederá a implementar algunas arquitecturas recurrentes con el fin de probar su efectividad a la hora de resolver la tarea. Para ello, se deberá seleccionar un dataset adecuado y se deberán aplicar modificaciones sobre dichos datos. Por último se detallarán las arquitecturas a probar así como los resultados obtenidos de cada experimento. El objetivo será por tanto generar pistas de audio melódicas que resulten agradables e (idealmente) indistinguibles de aquellas compuestas por un humano.

1.2 Estructura del documento

Este documento se estructura de la siguiente manera: En el capítulo 2 se revisará el estado del arte sobre la composición musical automática, destacando las principales investigaciones que se han llevado a cabo. En el capítulo 3 se detallarán los modelos a implementar, deteniéndose en el apartado de preparación de datos así como en la topología de las redes. El capítulo 4 cubrirá los experimentos realizados, comentando las pruebas para cada arquitectura así como los resultados obtenidos. Por último, en el capítulo 5 se concluirá el trabajo realizado, valorando los conceptos aprendidos, comparando las arquitecturas en base a los experimentos realizados y señalando algunas líneas de trabajo para proseguir la investigación.

2 Estado del arte

A lo largo de la historia de la computación, siempre ha existido un afán por el diseño de técnicas o sistemas que sean capaces de componer música de forma autónoma. Cuando se nombran este tipo de soluciones, se tiende intuitivamente a pensar en aplicaciones relacionadas con la Inteligencia Artificial, es decir, algoritmos con un gran componente probabilístico que generan por tanto soluciones no deterministas. Sin embargo, en épocas anteriores a la aparición de los primeros computadores, se pueden encontrar ejemplos de algoritmos clásicos que trabajan en el campo de la composición musical.

Uno de los primeros logros en la historia de la composición algorítmica apareció en el año 1024 de la mano de Guido de Arezzo, inventor del solfeo, quien propuso un método para convertir escritos religiosos en frases melódicas mediante una correspondencia texto - tono [40].

En el año 1650, Athanasius Kircher escribe *Musurgia Universalis* [27][40], donde presenta *Arca Musurgica*, una máquina capaz de generar música mediante el ajuste de una serie de parámetros por parte de la persona que la manejara.

Posteriormente, en el siglo XVII, un juego llamado *musical dice game* se popularizó en Europa [40]. Inicialmente creado por Johann Philipp Kirnberger y popularizado en 1972 por Mozart, el juego consistía en hacer tiradas con unos dados. En cada tirada, el jugador debía buscar en una tabla el fragmento musical que se correspondía con los números obtenidos y de esta forma ir construyendo la partitura. En cierto modo, este sería el primer algoritmo de carácter probabilístico con el que se generara música.

Como puede verse, los intentos de crear sistemas que permitan automatizar el proceso de creación musical han estado presentes desde hace siglos. Sin embargo, la aparición de la computación y los consecutivos avances tecnológicos han permitido el desarrollo de sistemas cada vez más complejos y creativos a la hora de generar música de manera autónoma. A continuación se va a hacer una revisión de dichas propuestas, haciendo

hincapié en las tecnologías que las soportan y revisando de manera detallada los modelos *Deep Learning* con los que se trabajará en este estudio.

2.1 Composición musical por computador

Como se ha mencionado con anterioridad, la aparición de los computadores supuso una revolución en la mayor parte de las ramas de la ciencia así como en la sociedad. La transversalidad de los computadores permitió que esta revolución tecnológica llegase a todo tipo de campos, en los que se empezaron a aplicar estas técnicas para crear nuevas ideas. Una de las áreas en las que se puede ver reflejada esta tendencia es la de la composición musical. Desde herramientas de apoyo al compositor hasta sistemas capaces de mostrar su propia "creatividad", la historia de la música desde la aparición de los computadores ha cambiado drásticamente.

En 1957, el ingeniero Max Mathews diseña en los laboratorios Bell el primer programa sintetizador conocido como *Music*, con el cual el autor Newman Guttman crearía la melodía "The Silver Scale" [4]. En palabras del propio Mathews: "Este software de procesamiento genera un onda, una onda triangular, con las mismas características de subida y bajada. Podías especificar un tono, una amplitud y una duración por cada nota y eso era todo." [43].

Ese mismo año, los compositores y profesores en la Universidad de Illinois en Urbana-Champaign Lejaren Hiller y Leonard Issacson programaron el sistema *ILLLIAC I* [17] [4]. Este sistema fue el primer algoritmo computacional en componer una partitura para un cuarteto de cuerda llamada *Illiac Suite*. Esta obra estaba formada por cuatro movimientos resultado del cómputo de cuatro experimentos con el sistema, involucrando desde técnicas como elecciones basadas en reglas, hasta modelos estocásticos como son las cadenas de Markov.

Iannis Xenakis, en 1962 presentan *Atrées*, un sistema de composición estocástica (como el ejemplo de *musical dice game* del apartado anterior) que seleccionaba las notas en función de un conjunto de probabilidades preestablecidas, generando así obras musicales.

En 1968, el anteriormente mentado Max Mathews continuando su carrera de investigación en el ámbito de la síntesis sonora y la interacción hombre - ordenador, hizo uso de *Graphic I*, desarrollado por William Nike. Según Max: "*El sistema Graphic I permite*

introducir imágenes y gráficos directamente a la memoria de un ordenador mediante el propio acto de dibujarlas." [20]. Max hizo uso de *Graphic I* como front-end para su sistema *Music IV*.

En esta misma línea, en 1970 Max, junto con F.R. Moore desarrollaron *GROOVE* [35]. Un sistema que mediante la interacción humana, permitía trabajar con hasta 14 ondas. Como puede verse en la Figura 2.1, el operario interactúa con el sistema por medio de una interfaz conformada por distintos *joysticks*, consiguiendo así jugar con las distintas ondas. Este programa permitía trabajar en diferentes dominios, sin embargo como se explicita en el artículo anteriormente referenciado: "*Pese a que GROOVE es un programa de propósito general, ha sido usado inicialmente para controlar un sintetizador electrónico de música formado por osciladores cuya frecuencia es establecida por voltaje, y amplificadores, cuya ganancia es igualmente establecida mediante voltaje*".

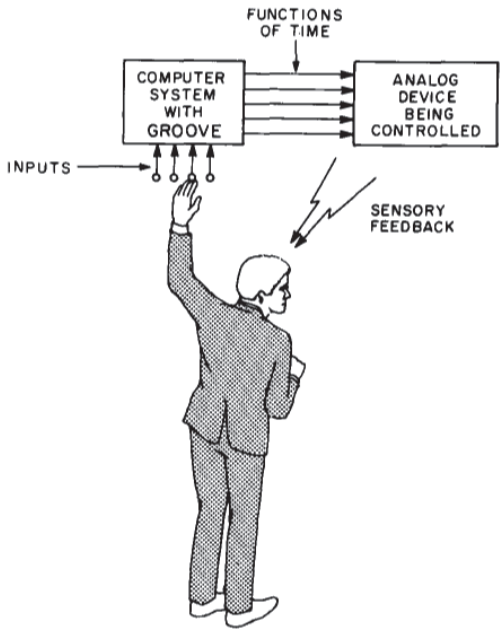

FIG. 1. Feedback loop for composing functions of time

Figura 2.1: Humano interactuando con GROOVE [35].

Estos sistemas sentaron las bases de muchos de los avances que conocemos hoy en día. Si se atiende a los ejemplos vistos, se pueden observar dos tendencias: sistemas que ayudan a la composición y sistemas que crean musica de manera autónoma.

En el primer grupo podemos encontrar, por ejemplo los sintetizadores. Estos instrumentos permiten crear infinidad de sonidos mediante la modificación de las propias ondas, todo esto guiado a través de una interfaz gráfica. Ejemplos de estos sintetizadores

en el panorama musical pueden encontrarse sobre todo desde la musica de los años 70 hasta la actualidad, donde es difícil encontrar canciones que no incluyan este tipo de herramientas. También otro tipo de software englobado en este grupo son los llamados asistentes de composición. Este tipo de herramientas sugieren al compositor progresiones de acordes y diferentes notas en función de un estilo en concreto o una partitura dada como referencia.

El otro grupo y en el que se centra este trabajo, es el de los algoritmos de composición autónoma. Con el avance de la tecnología y la aparición de nuevos modelos matemáticos y computacionales, este tipo de sistemas han dado un gran salto cualitativo. Más adelante se detallarán dichos modelos, haciendo hincapié en uno de ellos, las redes neuronales recurrentes.

Antes de entrar en detalle con los diferentes modelos para la composición automática de música, es necesario repasar con qué datos se trabaja en este campo y cómo se han de representar para poder ser utilizados por dichos modelos.

2.2 Tipos de datos y su representación

En esta sección se aborda una de las partes clave en toda aplicación informática: los datos. Una de las tareas fundamentales a la hora de realizar el diseño de un sistema consiste en entender los datos con los que trabaja, saber qué características tienen y por tanto establecer una forma adecuada de representarlos.

En el caso de los sistemas de composición automática, es necesario conocer cómo puede representarse de manera formal la música. Se ha de distinguir entre sistemas que puedan procesar audio o sistemas que trabajen con representaciones simbólicas. Se ha de seleccionar una representación que permita capturar la mayor cantidad de características posible, de modo que el modelo a entrenar pueda generar obras expresivas y cercanas a la realidad. Es precisamente esta elección a la hora de representar los datos la que nos determinará qué tipo de arquitecturas y modelos podrán trabajar con ellos.

Antes de ver los diferentes tipos de representación de la información musical, es conveniente repasar algunas características básicas de la música, de modo que se entienda qué es lo que se debe representar.

2.2.1 Conceptos básicos

Atendiendo a su definición, la música consiste en la organización lógica de sonidos en de manera que exista una coherencia melódica, armónica y rítmica. A lo largo de la historia, la música se ha representado de diversas formas con el fin de poder trabajar con ella. Es decir, pese a que la música en si es un conjunto de ondas sonoras, para trabajar con ella se ha de dar una representación simbólica. En ese sentido existe un vocabulario estándar sujeto a ciertas reglas que se debe usar para crear música.

Nota

El elemento fundamental en la representación musical es la nota. En la notación musical, existen doce posibles notas, cada una de ellas representando un sonido en concreto. Al igual que el resto de elementos del lenguaje musical, las notas se escriben en partituras.

Figura 2.2: Nota de sol (G) natural representada en una partitura.

- **Tono**: El tono es la propiedad que permite distinguir unas notas de otras unívocamente. Esta propiedad se refiere a la frecuencia de vibración de la onda. En notación americana, las diferentes notas se representan con letras del abecedario de la A a la G. Esta notación será la usada en la gran mayoría de modelos de composición automática.
- **Duración**: La duración de la nota hace referencia al tiempo que esta está sonando. Típicamente, la duración base corresponde a 4 tiempos, siendo esta la redonda. El resto de notas son subdivisiones (blanca, negra, corchea, etc.).
- **Intensidad**: Se refiere precisamente a la intensidad con la que se interpreta dicha nota en el instrumento. Esta propiedad dota de expresividad a la reproducción de la nota.

Cuando dos notas son tocadas a la vez (u ocasionalmente en secuencia), se habla de un intervalo. Dependiendo de la relación tonal entre dichas notas, el intervalo podrá ser

de segunda, de tercera, de quinta perfecta, etc. Los intervalos son un concepto importante y un punto inicial para la construcción de acordes.

Existe un tipo especial de notas, las cuales solo tienen la propiedad de duración. Se conocen como silencios y su función es precisamente ocupar tiempo sin reproducir ningún sonido.

Acordes

Los acordes conforman el elemento armónico dentro de la música. Se trata de un conjunto de, como mínimo, tres notas que se reproducen al mismo tiempo, como puede verse en la Figura 2.3.

Figura 2.3: Acorde de sol (G) mayor.

Al igual que las notas, es frecuente ver los acordes representados mediante notación americana. En este caso el acorde se representa por la letra de su nota principal o tónica. Adicionalmente se añade una palabra que clarifica el tipo de acorde si este añade una o más notas, por ejemplo: Gmaj7 (acorde de sol mayor con séptima mayor), Gmin7 (acorde de sol menor séptima), etc. De nuevo, esta representación será útil de cara a presentar este tipo de datos a los modelos.

Ritmo

El último punto de este repaso y uno de los más importantes en la música es el ritmo. El ritmo marca cómo se divide y estructura la obra. Juega un papel fundamental en la expresividad de la musica a generar, por lo que debe ser estudiado.

El elemento básico del ritmo dentro de un compás se conoce como pulso. Los compases dividen la partitura en segmentos de igual duración e igual número de pulsos. La métrica determina precisamente la duración de los compases, siendo las métricas más comunes 2/4 (cada compás dura dos negras), 4/4 (cada compás dura cuatro negras) y 3/4 (cada compás dura tres negras). La velocidad a la que se debe interpretar la partitura viene dada por un valor llamado pulsos por minuto o *beats per minute (bpm)*.

Una vez se han visto los elementos básicos mediante los cuales se escribe la música, es el momento de repasar qué tipos de representación de la música encontramos en el ámbito de los sistemas informáticos.

2.2.2 Representación en audio

Como se ha expuesto anteriormente, la música puede entenderse como un conjunto de ondas, es decir como cualquier otro tipo de sonido. Como tal, la forma más directa de representar la música es haciendo uso de su propia onda de audio.

Onda de audio

Figura 2.4: Pista de audio generada mediante una guitarra eléctrica.

Como puede verse en la Figura 2.4, en este formato se presenta la amplitud de la onda en el eje vertical a lo largo del tiempo representado en el eje horizontal.

Existen ejemplos de sistemas de composición musical que utilizan este tipo de representación. Por ejemplo, el trabajo propuesto por Nayebi y Vitelli [39], en el que entrenan una arquitectura basada en redes neuronales recurrentes LSTM y GRU (ver Sección 2.3.3) haciendo uso de datos en formato de onda de audio. Otro ejemplo que hace uso de redes neuronales convolucionales (ver Sección 2.3.1) para generar audio en base a ejemplos en formato de onda de audio es el sistema *Wavenet* [41].

Como Briot *et al.* exponen en su artículo [4], el problema de esta representación es la cantidad de recursos computacionales necesarios para que los sistemas que trabajan con ella puedan rendir adecuadamente. Por ello, en dicho artículo los autores hacen referencia otra forma de representación derivada de ésta: el espectrograma.

Espectrograma

Un espectrograma es la representación de una onda obtenida tras su descomposición armónica mediante la *transformada de Fourier*. En un espectrograma correspondiente a una onda acústica, la señal original es separada en cada uno de los armónicos que la componen, mostrando así información relativa a las diferentes frecuencias de manera individual.

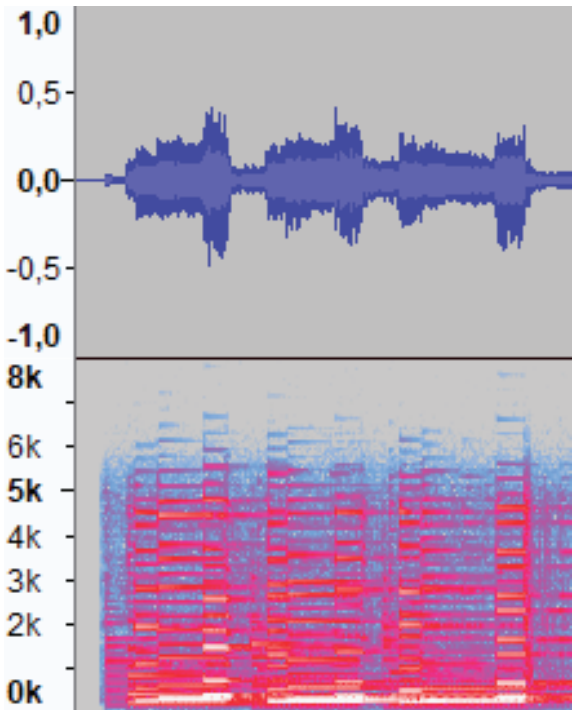

Figura 2.5: Espectrograma correspondiente a una pista de audio generada mediante una guitarra eléctrica.

En la Figura 2.5 puede verse el espectrograma correspondiente a la onda de la Figura 2.4. En el eje vertical, se representan los diferentes rangos de frecuencia en kHz. En el eje horizontal se representa el tiempo en segundos. Por último, mediante una escala de colores, se refleja la intensidad de la onda en decibelios.

Más adelante, se hablará de un tipo concreto de redes neuronales que pueden ser usadas para generar música de manera automática haciendo uso de este tipo de datos (ver Sección 2.3.1).

2.2.3 Representación simbólica

Mientras que la representación mediante audio podía entenderse como una codificación en la que notas, acordes y ritmo estaban presentes de manera implícita, la representación simbólica hace uso directo de los elementos vistos en la Sección 2.2.1. El medio clásico y más relevante para plasmar de manera simbólica información musical es la partitura. Mediante una partitura, pueden expresarse todos los detalles de una composición de manera precisa y además pueden incluirse datos como el autor, el año, etc. Sin embargo, de cara al diseño de sistemas de composición autónoma, existen representaciones más efectivas.

MIDI

Dentro de esta categoría, el formato más usado para el ámbito de las aplicaciones informáticas es el MIDI (*Musical Instrument Digital Interface)* [37]. Se trata de un estándar que propone un protocolo, una interfaz y unos conectores hardware que permiten la comunicación entre ciertos instrumentos y un ordenador.

Esta tecnología se basa en eventos para manejar la información musical. Estos mensajes pueden ser de tres tipos: eventos de información musical (tono, notación, intensidad), eventos de control (dinámica, vibrato, etc.) y eventos de sincronización. Los más relevantes de cara extraer características para sistemas informáticos son los eventos de información. En concreto, el mensaje más útil es el que porta información sobre cuándo una nota comienza a ser tocada y cuando se detiene.

- **Note on**: Este mensaje se envía cuando una nota es tocada. En el mensaje, se representa mediante un entero entre 1 y 127 el tono de la nota, incluyendo en este rango las 12 notas posibles en diferentes octavas. Además, el mensaje contiene información sobre la velocidad de la nota, también entre 1 y 127. Por último, dado que en MIDI es posible representar hasta 15 pistas de audio con diferentes instrumentos cada una, en el mensaje se incluye el número de pista.
- **Note off**: La contra parte del mensaje anterior indica cuando la nota deja de ser tocada. De igual manera, se incluye el número de pista, el tono de la nota y la velocidad a la que se libera la nota.

Cada uno de estos mensajes se engloba en fragmentos de pista, que portan información temporal que permite localizar el evento en el conjunto global de la obra.

Como se ha explicado, los archivos MIDI tienen la capacidad de albergar hasta 15 pistas de audio. Cada una de ellas puede corresponderse con un instrumento diferente, de modo que pueden codificarse todo tipo de géneros musicales. Es posible separar y tratar individualmente cada pista de audio, esto supone una ventaja a la hora de desarrollar sistemas de composición autónoma, ya que es posible entrenar modelos que aprendan a generar diferentes instrumentos haciendo uso de un mismo archivo MIDI [32] [58].

Al tratarse de un estándar, existen una gran cantidad de recursos musicales en formato MIDI en internet. De cara al desarrollo de sistemas de composición autónoma, pueden encontrarse datasets de una gran variedad de géneros musicales [11]. También hay datasets con multitud de pistas de audio MIDI de un único instrumento, en el caso de que lo que se busque sea un sistema que elabore un tipo de voz en concreto [15]. Además, por toda la red existen numerosas aportaciones de usuarios que recopilan datasets en función de la necesidad [48].

Cabe señalar que este formato puede ser usado simplemente como fuente de datos. Como se acaba de ver, dada la gran cantidad de recursos online es facil encontrar un dataset de archivos MIDI que se ajuste a las necesidades del problema. Una vez obtenidos los datos, se hace uso de una de las múltiples librerías de tratamiento de MIDI para extraer la información musical.

Piano roll

Tradicionalmente, el *piano roll* ha servido como medio de almacenamiento de información musical. Se trata de una cinta perforada que hace funcionar un tipo especial de piano: la pianola (o piano mecánico). Cada perforación representa una nota y a medida que la cinta va moviéndose por la pianola, una barra detecta los huecos y mediante una serie de mecanismos producen el sonido. Sin embargo, el *piano roll* físico ha sido adaptado para su uso en sistemas digitales.

En la Figura 2.6 se ve un ejemplo de esta representación usada en muchos programas de procesamiento de audio y composición. Sin embargo, como señalan Briot et al. [4], el *piano roll* tiene ciertas limitaciones. Por ejemplo, al contrario que pasa en MIDI, en *piano roll* no existe información sobre cuándo una nota deja de ser tocada, de modo que es indistinguible cuando suena una nota larga, o dos cortas repetidas. Los mismos autores recogen algunas técnicas para solventar este problema.

Esta representación es muy usada en la literatura. Por ejemplo, puede verse su uso en

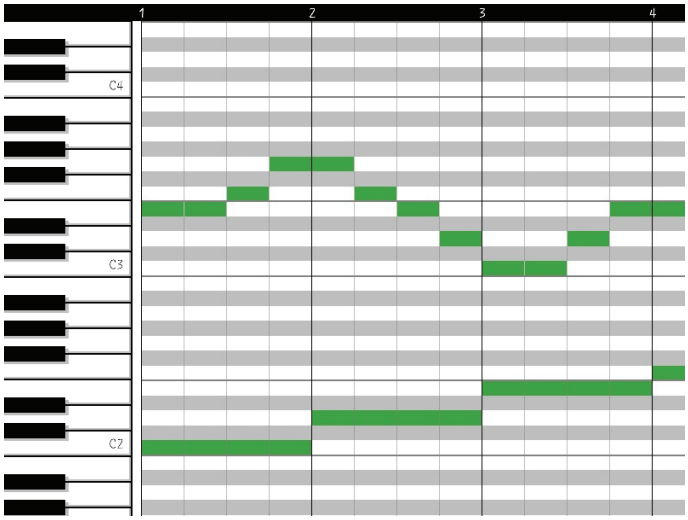

Figura 2.6: Ejemplo melodía en formato *piano roll* [4].

el sistema propuesto por Daniel Johns [24] en su blog *Hexaedria*, donde el autor añade información extra sobre el vecindario de cada nota (notas previas y posteriores) así como sobre el ritmo. Otro de los numerosos ejemplos se puede encontrar en el estudio de Todd y Peter [51], aplicado a la composición automática con redes neuronales recurrentes.

Texto en notación americana

Como se ha mencionado con anterioridad, la notación americana es una forma muy frecuente de representar obras musicales, sobre todo de musica moderna (jazz, rock, neo-soul, etc.). Se trata de una representación muy directa y sencilla de usar en sistemas informáticos.

La principal ventaja de este tipo de representación es que de cara a los modelos de *Deep Learning*, puede tratarse como si fuese información textual. Esto implica que ciertos modelos diseñados para la generación automática de texto, pueden ser usados para generar música representada en este formato.

Mediante las técnicas de representación por notación americana, es posible especificar notas y acordes haciendo uso del alfabeto. Sin embargo, para expresar una nota, es necesario usar más de un carácter, ya que se precisa información aparte del tono expresado con la letra. Por ejemplo, es necesario saber si es natural, sostenida o bemol; la octava en la que está siendo tocada, etc. En el caso de los acordes, se necesita como mínimo, tres letras para representar las tres notas (mas el resto de elementos ya comentados).

Una de las posibles variantes de este tipo de codificación es, la notación ABC [53]. Un

ejemplo de su uso puede verse en trabajos como el de Sturm *et al.*, en el que se emplea esta representación, pero añadiendo la agrupación de caracteres en *tokens* para representar de manera adecuada las características de las obras [50].

2.2.4 Codificación

Una vez se ha escogido convenientemente cuál va a ser el formato en el que se representarán los datos, es el momento de transformar dicha información en valores numéricos de manera que los modelos que se usen puedan trabajar con ella. Dependiendo de la representación de los datos, las estrategias de codificación podrán variar.

Por ejemplo, en caso de utilizar una representación subsimbólica para modelos convolucionales, no será necesario nada más que presentar el audio en forma de imagen a la red [41]. También existen trabajos en los que se usan estas representaciones en redes neuronales recurrentes, mediante la división del audio en intervalos temporales anotando los valores de la onda como entrada de la red [39].

En el caso de trabajar con representaciones simbólicas, como es el caso de la mayoría de las propuestas en el ámbito del *Deep Learning*, se deben aplicar técnicas específicas para transformar los datos en valores numéricos interpretables por los modelos. En su articulo, Briot *et al.* proponen una clasificación de posibles métodos de codificación para este tipo de datos [4]:

- **Valor numérico continuo**: Este método consiste en tomar el valor de la frecuencia en Hz de la nota (el tono) para representarla unívocamente. El dominio en este caso serían los números reales.
- **Valor numérico discreto**: La segunda opción consiste en hacer uso del valor que el sistema MIDI le otorga a la nota en función de su tono.
- ***One-hot encoding***: Esta técnica tradicional de codificación consiste en representar los datos mediante un vector de dimensión igual al número de elementos diferentes que existen (vocabulario). Todos los elementos del vector estarán a 0 excepto aquel que represente a la nota en cuestión, que estará a 1. Esta estrategia es apropiada para lineas monofónicas, sin embargo para cuando hay varias notas o pistas sonando a la vez, los autores proponen las siguientes opciones.
- ***Many-hot encoding***: Es igual que *one-hot encoding* pero en este caso todas las notas que están siendo tocadas se representan con un 1 en el vector.

- ***Multi-one-hot encoding***: Se trata de usar un vector con formato *one-hot encoding* para representar cada una de las voces o pistas sonando en ese momento.
- ***Multi-many-hot encoding***: Se trata de usar un vector con formato *many-hot encoding* para representar cada una de las voces o pistas sonando en ese momento. Se utiliza cuando al menos una de las pistas de audio tiene varias notas sonando a la vez.

Como puede verse, cuando se trata con representación de acordes, lo más adecuado es usar estrategias del tipo *many-hot*, puesto que varias notas suenan a la vez. Sin embargo, el uso de este tipo de codificaciones puede resultar en vectores de gran dimensionalidad con muchos ceros, es decir, *sparse vectors*.

Además, mediante este tipo de codificaciones, no se captura las relaciones entre elementos. Por ejemplo, un musico puede reconocer que por motivos de tonalidad, un acorde de do mayor está más relacionado con uno de re menor que con uno de re semidisminuido. En el campo del Procesamiento del Lenguaje Natural se presenta el mismo problema con la relación semántica entre términos, y es solventado mediante los llamados *embeddings*. Los *embeddings* son representaciones vectoriales de los datos en las cuales aquellos términos relacionados semánticamente también los están numéricamente. En el procesamiento de información musical, Madjiheurem *et al.* propusieron un método de *embeddings* para acordes: Chord2Vec [34].

2.3 Deep Learning en la composición musical automática

En la sección anterior, se ha repasado los diferentes tipos de datos con los cuales se trabaja en el dominio de la generación musical automática. Una vez se han recopilado los datos necesarios y se han procesado de manera adecuada, es el momento de utilizarlos como entrada para los diferentes modelos generativos.

Dentro del campo de estudio de la inteligencia artificial, hay un gran número de algoritmos y modelos propuestos para resolver problemas de aprendizaje automático. Sin embargo, uno de ellos cobra cierto protagonismo respecto al resto tanto para la opinión pública como para la comunidad científica: las Redes de Neuronas Artificiales. En concreto, a la hora de desarrollar tareas con cierto carácter creativo, existen ciertas arquitecturas de *Deep Learning* que destacan por encima del resto.

En esta sección, se va a hacer un repaso de las principales arquitecturas de redes neuronales que típicamente se usan para atajar problemas de generación automática de música.

2.3.1 Redes Neuronales Convolucionales

Las redes neuronales convolucionales o *Convolutional Neural Networks (CNN)* son un tipo concreto de red neuronal profunda. Su principal característica es que en lugar de usar multiplicaciones de matrices comunes, utiliza en al menos una de sus capas, la operación matemática conocida como convolución [13].

Esta arquitectura surgió de la mano de LeCun, quien adaptó la operación de convolución a las redes neuronales para el reconocimiento de dígitos escritos a mano [31]. Posteriormente, el propio LeCun perfeccionó el diseño para aplicarlo a imágenes [30].

En la Figura 2.7 puede verse un ejemplo de este tipo de modelos. A la entrada, se presenta una determinada imagen, sobre la que se aplican filtros y se ejecuta la operación de convolución. Puede haber varias capas convolucionales mediante las cuales se van extrayendo rasgos concretos de las imágenes (lineas, curvas, etc). Posteriormente, es común situar algunas capas densas, donde se lleve a cabo el resto del procesamiento de la arquitectura. Por ejemplo, en el caso de una arquitectura de clasificación, estas capas densas terminarían con una capa de salida con tantos nodos como clases existan y con función de activación como por ejemplo softmax.

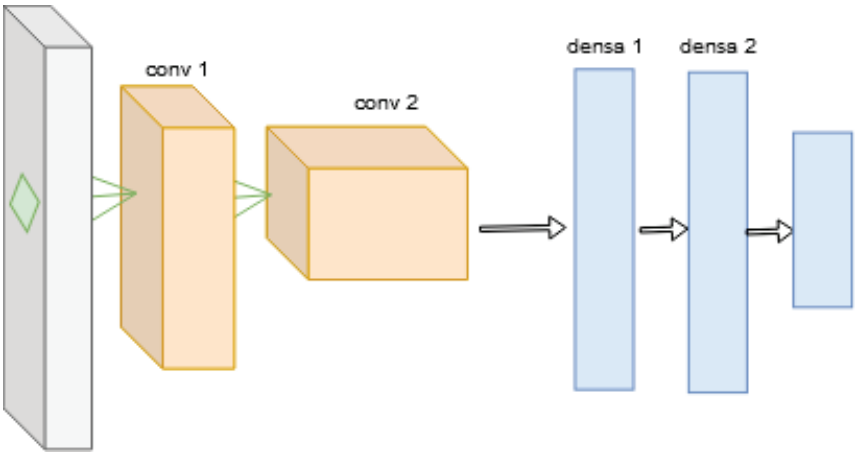

Figura 2.7: Arquitectura general de una red convolucional.

En 2012, Alex Krizhevsky presentó *AlexNet* [28], una red neuronal convolucional que consiguió un top 5 en *ImageNet Large Scale Visual Recognition Challenge*, un *benchmark*

de clasificación de objetos en más de cien categorías y con millones de imágenes [46]. Todo esto resultó en uno de los modelos de redes neuronales más usados de la actualidad y el principal exponente en el área de la visión por computador.

Sin embargo, este tipo de red es susceptible de ser usada para resolver problemas que no estén necesariamente dentro del campo de la visión por computador. Es decir, si los datos se pueden representar mediante imágenes, este tipo de arquitectura puede ser una opción a considerar. En concreto, a la hora de abordar el problema de la composición automática de música, este tipo de redes neuronales ha sido usada en ciertas ocasiones.

El ejemplo más destacado es *Wavenet*. Este sistema fue presentado por Oord *et al.* en 2016 [41]. Se trata de un modelo basado en redes neuronales convolucionales que trabaja con audio en crudo (ver Sección2.2.2) como entrada y lo produce como salida. En el artículo se expone que la arquitectura se ha probado para resolver cuatro problemas: reconocimiento del habla (*speech recognition*), texto a habla (*text-to-speech*), generación multi hablante (*multi-speeker speech generation*) y por último, generación musical. Su principal característica es el uso de capas convolucionales causales con el fin de obtener percepción de patrones en steps anteriores. Los autores comentan que en los experimentos probaron con dos datasets musicales. El primero de ellos formado por 200 horas de audio, dividida en clips de 29 segundos, los cuales eran etiquetados por género, instrumentos, etc. El otro dataset, formado por 60 horas de música de piano de YouTube. Los resultados fueron coherentes armónicamente pero fallaban en conseguir reproducir patrones a lo largo del tiempo, cambiando de género, volumen y calidad cada pocos segundos.

2.3.2 Redes Neuronales Generativas Adversarias

Las redes Neuronales Generativas Adversarias o *Generative Adversarial Networks (GAN)* fueron propuestas en 2014 por Goodfellow *et. al* [14]. La principal característica de este modelo es que combina dos redes neuronales, siendo una de ellas la responsable de crear valores de salida y la otra responsable de evaluar la calidad de dichas salidas.

Como puede verse en la Figura 2.8, la red generadora (*Generator*) toma como entrada una variable de ruido aleatorio y ofrece como salida un ejemplo que debe ser lo más parecido posible al dataset de entrenamiento. Por otra parte, la red discriminante (*Discriminator*) recibe ejemplos tanto del generador como del dataset de entrenamiento, debiendo estimar la probabilidad de que los datos recibidos pertenezcan al dataset ori-

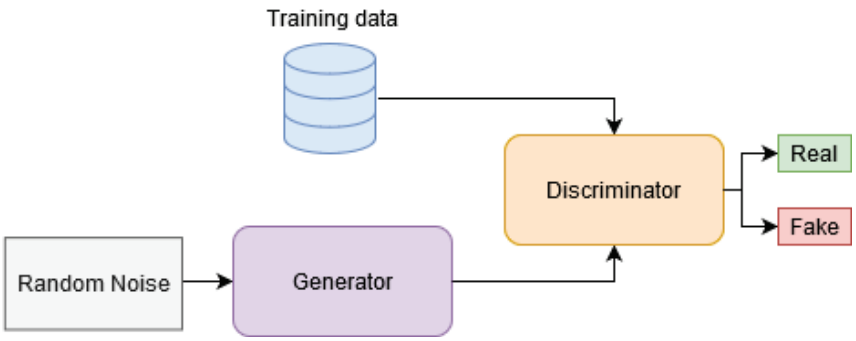

Figura 2.8: Arquitectura GAN.

ginal. El modelo se considera bien entrenado cuando el generador es capaz de devolver ejemplos de manera que el discriminante no sabe distinguir y evalúa como procedentes del dataset de entrenamiento.

El éxito de esta arquitectura puede verse reflejado sobre todo en sistemas de generación de imágenes. En este campo cabe destacar *StyleGAN*, un método de generación de imágenes de alta resolución de rostros humanos propuesto por Karras *et al.* [25] [26].

Sin embargo, también existen aportaciones en el uso de esta arquitectura al campo de la composición musical. Un ejemplo es el sistema *MidiNet*, el cual utiliza arquitecturas convolucionales para las redes generadora y discriminante. Este sistema hace uso de 1022 canciones pop en formato tablatura (similar al formato *piano roll*), teniendo cada tablatura una linea de acordes y otra de melodía. Existe por tanto una versión del sistema con tan solo generación de melodía y otra que se apoya en la progresión de acordes.

Otro ejemplo es el sistema *Musegan*, creado por Dong *et al.* [9]. Los autores utilizan un dataset de canciones rock conteniendo múltiples pistas instrumentales (guitarra, batería, bajo, etc.) para el entrenamiento de tres modelos: modelo de *jamming*, modelo de composición y modelo híbrido. Finalmente, *MuseGAN* se constituye como la integración y extensión de dichos modelos.

2.3.3 Redes Neuronales Recurrentes

Hasta el momento, se han revisado dos modelos que han sido usados en numerosos trabajos en el campo de la generación musical automática. Sin embargo, si hay un tipo de redes neuronales que destaca por encima del resto son las redes neuronales recurrentes.

En este capítulo se va a realizar una revisión de las redes neuronales recurrentes [12]. En primer lugar se realizará una introducción general, explicando cómo se compone

la red, qué tipos de topologías se pueden encontrar y qué problemas se derivan de las celdas de memoria más básicas. A continuación se expondrán dos celdas de memoria más complejas, las redes *Long Short-Term Memory* (LSTM) y las redes tipo *Gated Recurrent Unit* (GRU).

Introducción a las redes neuronales recurrentes

Uno de los comportamientos que nos caracterizan como humanos es la habilidad para poder predecir acontecimientos que sucederán en el futuro. Por ejemplo, si vamos por la calle y vemos como un coche se acerca, somos capaces de determinar en qué momento llegaría a colisionar con nosotros atendiendo a la trayectoria que está siguiendo. En ciertos casos, es deseable que este comportamiento tan humano pueda ser reproducible por medio de sistemas informáticos. De este modo, se podrían realizar tareas de predicción siguiendo la tendencia de ciertos valores a lo largo del tiempo.

En 1982, John Hopfield, basándose en los trabajos previos de David Rumelhart [45], ideo lo que sería la primera versión de neurona recurrente, la red de Hopfield [21]. Las redes neuronales recurrentes (*Recurrent Neural Networks* - RNN) tienen la capacidad de analizar secuencias temporales de datos de tamaño variable y predecir cuál será el siguiente valor en la serie. Este tipo de comportamiento es, en cierto modo, un medio para obtener originalidad y creatividad. En el caso de la composición automática musical, las redes neuronales recurrentes toman como entrada una serie temporal de notas y predicen con cierta probabilidad, cual será la siguiente nota.

Desde un punto de vista general, las redes neuronales recurrentes pueden considerarse un caso específico de redes neuronales convencionales en las que se añade una conexión extra desde la salida de la neurona hacia la propia entrada. Como puede verse en la Figura 2.9 cada unidad recibe los datos de entrada, produce una salida y envía dicha salida de nuevo hacia sí misma. Por tanto, cada neurona recurrente tendrá un conjunto de pesos w_x para los datos de entrada, y otro conjunto de pesos w_y para la salida del time step anterior.

Un modo más intuitivo de ver el funcionamiento de este tipo de neuronas, es desarrollarlas a lo largo del tiempo. Como se ve en la Figura 2.9, en cada time step, la neurona recibe el dato de entrada correspondiente y además, la salida del time step anterior. Este funcionamiento permite a la red recordar información de secuencias anteriores a la hora de procesar los datos del time step actual.

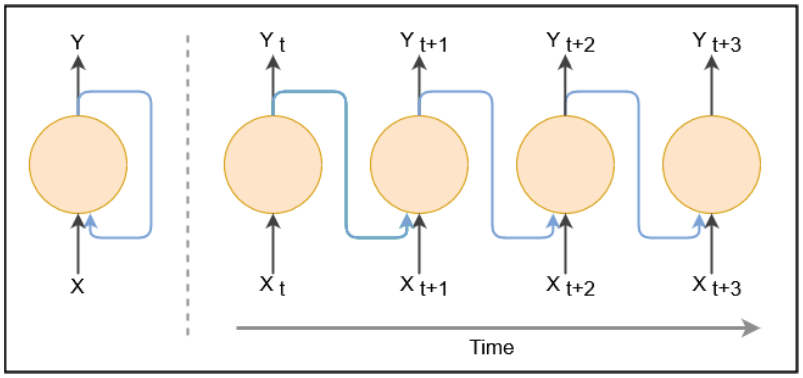

Figura 2.9: A la izquierda, neurona recurrente. A la derecha, neurona recurrente desarrollada en el tiempo.

De manera sencilla se pueden construir capas de neuronas recurrentes, como puede verse en la Figura 2.10.

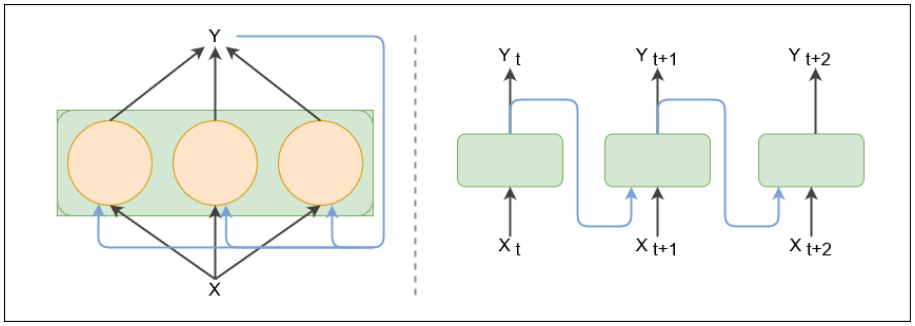

Figura 2.10: A la izquierda, capa recurrente. A la derecha, capa recurrente desarrollada en el tiempo.

En este caso, cada neurona recibirá el vector de entrada y el vector de salida correspondiente al valor devuelto por todas las neuronas de la capa en el time step anterior. Si se toman en cuenta los pesos de todas las neuronas de la capa, se pueden disponer matrices de pesos W_x para la entrada y W_y para la salida del time step anterior. En la Ecuación 2.1 puede verse el cómputo del valor de salida de la capa para una instancia dada.

$$y_{(t)} = f_{activation}({W_x}^T x_{(t)} + {W_y}^T y_{(t-1)} + b) \tag{2.1}$$

De este modo, el cómputo del valor de salida para cada time step t pueda realizarse de manera simultanea para todas las instancias del lote o *batch* que se esté procesando,

como puede verse en la Ecuación 2.2.

$$Y_{(t)} = f_{activation}(X_{(t)}W_x + Y_{(t-1)}W_y + b) \tag{2.2}$$

Si apilamos varias capas de neuronas recurrentes podemos hablar entonces de *deep RNN* o redes neuronales recurrentes profundas.

Una vez entendidas las capas de neuronas recurrentes, puede presentarse el concepto de *celda de memoria* [12]. Normalmente se usa este término para referirse a capas de neuronas recurrentes, ya que a efectos su funcionamiento se basa en "recordar" información de time steps pasados. Es decir, la salida de la capa en un time step concreto, viene determinada por los datos de entrada que la capa ha visto hasta ese momento. Bajo esta nomenclatura, se define la salida del time step anterior, $y_{(t-1)}$, como el estado de la celda de memoria, $h_{(t-1)}$. Se puede entender como una función de las entradas del time step actual y del estado de la celda del time step anterior: $h_{(t)} = f(h_{(t-1)}, x_t)$.

Uno de los primeros ejemplos de composición melódica haciendo uso de redes neuronales recurrentes clásicas es el sistema *CONCERT*, desarrollado por Mozer en 1994 [38]. El sistema fue entrenado con obras de *Bach*, representando la información musical mediante la codificación del tono, el tempo y el acompañamiento musical (acordes). El tono fue representado mediante un escalar representando su altura, las coordenadas en el circulo cromático y las coordenadas en el circulo de quitas. El tempo, subdividiendo la duración de una negra en 12, permitiendo así representar compases binarios y ternarios. Por último, los acordes fueron representados por medio de triadas y tetradas. Para obtener las otras de salida en la fase de generación, se seguía una distribución probabilística, de modo que este modelo tenía un comportamiento no determinista. El principal problema de esta propuesta era que la red no era capaz de retener patrones a largo plazo y por lo tanto, no había secuencias musicales coherentes. Esto es debido al problema del que acaecen las redes neuronales recurrentes (ver Sección 2.3.3).

Topologías en el campo de las redes neuronales recurrentes

Dependiendo del problema que se trate de resolver, la red podrá trabajar con diferentes tipos de secuencias de entrada y de salida [12]. Por ejemplo, podemos estar ante un problema de análisis de series temporales en un mercado, en el que la red deberá tomar como entrada el valor de ciertas acciones en los últimos n días y predecir cual será su valor en el día $n + 1$. En otro caso puede desearse predecir los valores para cada uno de

los días, basándose en el valor del día anterior. Entre otros, estos dos ejemplos dan una perspectiva de lo que se puede conseguir con redes neuronales recurrentes si se elige la topología adecuada. Se pueden definir cuatro topologías principales: *sequence-vector*, *sequence-sequence*, *vector-sequence*, *Encoder-Decoder*.

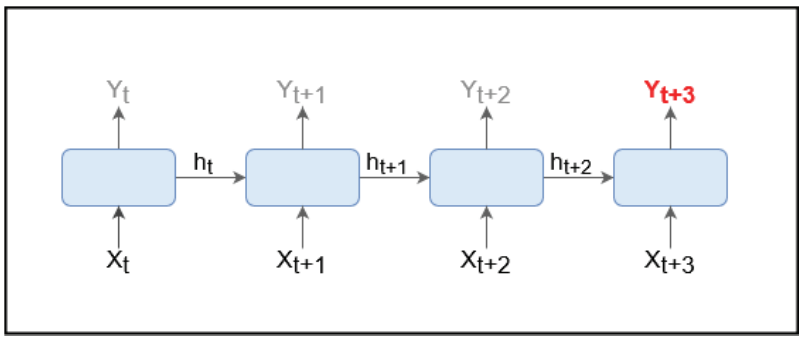

Figura 2.11: Topología tipo sequence-vector.

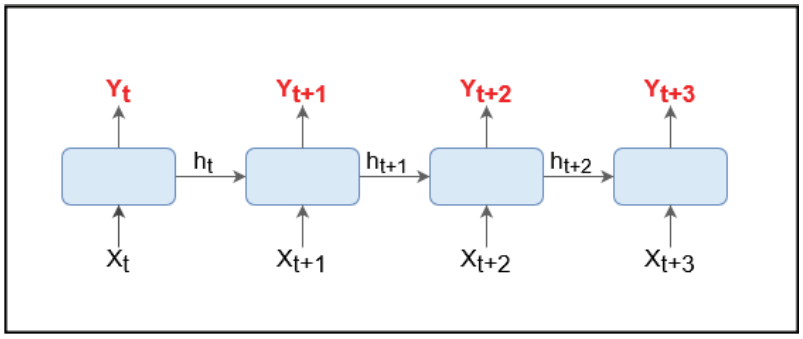

Figura 2.12: Topología tipo sequence-sequence.

La Figura 2.11 muestra la topología *sequence-vector*. La imagen muestra una capa de neuronas recurrentes desarrollada a lo largo del tiempo en la que sólo la salida del último time step es tomada en cuenta como salida de la capa. En la Figura 2.12 se ve la topología *sequence-sequence*, donde la capa devuelve el resultado procesado tras cada time step.

Usualmente, las redes neuronales recurrentes profundas que buscan un comportamiento tipo *sequence-vector* se diseñan acoplando varias capas tipo *sequence-sequence* y una última capa de tipo *sequence-vector*. Como consecuencia de esta estructura, la salida de la red será un único elemento que se calculará en función del resto de time steps del lote.

Otra de las topologías clásicas se puede ver reflejada en la Figura 2.13. Esta implementación *vector-sequence* se caracteriza por recibir entradas cuya longitud de secuencia es de un único time step. La capa va devolviendo los resultados del procesamiento du-

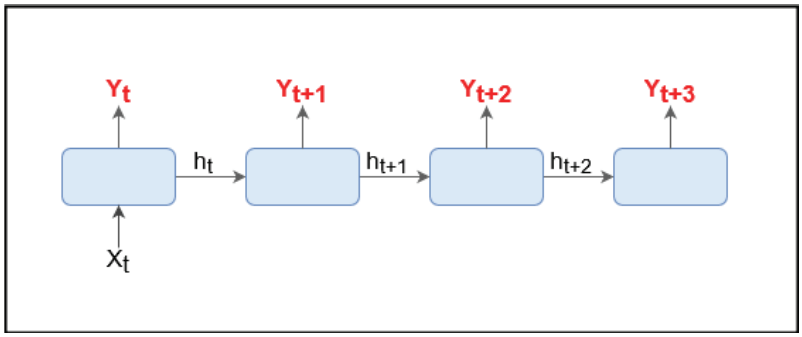

Figura 2.13: Topología tipo vector-sequence.

rante los n time steps designados, siendo la única entrada a la célula en los time steps posteriores al primero, el estado de la celda en el paso anterior.

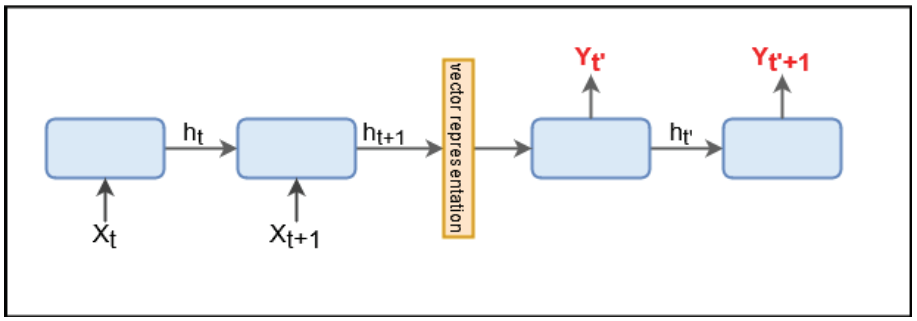

Figura 2.14: Topología tipo Encode-Decoder.

La última de las arquitecturas, *Encoder-Decoder* [5], deriva de la unión de *sequence-vector* y *vector-sequence*. Como puede verse en la Figura 2.14, la primera parte del modelo, el *Encoder*, toma como entrada una secuencia y sin devolver ningún resultado, genera en última instancia una representación vectorial de la misma (modelo *sequence-vector*). La segunda parte del modelo, el *Decoder*, toma dicha representación vectorial como entrada $h_{(t)}$ y va devolviendo el resultado del procesamiento en cada time step. El modelo global puede verse como un *sequence-sequence* en el cual la longitud de la secuencia de entrada puede variar con respecto a la de la secuencia de salida. Esto es útil en problemas como la traducción automática, en el que una frase en un idioma determinado puede tener más símbolos o palabras que su homóloga en otro idioma diferente. Este tipo de modelos están teniendo una relevancia significativa en el campo del Procesamiento del Lenguaje Natural, sobre todo desde la aparición de los mecanismos de *Attention*, cuya inclusión en estos modelos resultó en las arquitecturas *Transformer* [52].

Este tipo de arquitectura ha sido usado también en el ámbito de la composición automática. En ese sentido, puede señalarse el trabajo de Adam Roberts *et al.* [44], quienes

propusieron una mejora a los *Variational Autoencoders (VAE)* de modo que lidiasen mejor con datos secuenciales. Este modelo era capaz de extraer en primera instancia *embeddings* de la secuencia de notas de entrada para posteriormente generar las secuencias de salida. La arquitectura estaba compuesta por una primera red neuronal recurrente bidireccional trabajando como *Encoder*, la cual tenía 2048 nodos de entrada y una capa oculta con 212 celdas LSTM. La segunda red, funcionando como *Decoder*, estaba formada por una red recurrente jerárquica en dos niveles: una primera red encargada de generar los *embeddings* y otra segunda red que los usaba como estado inicial para generar las secuencias.

El problema de las redes neuronales recurrentes

Hasta este momento hemos visto cómo funcionan las celdas de memoria más básicas, las celdas RNN típicas. Este tipo de estructuras tienen, sin embargo, un problema llamado *short-term memory* [19] [18]. Este problema deriva del bien conocido desvanecimiento del gradiente o *vanishing gradient problem*. Como se ha explicado hasta ahora, las redes neuronales recurrentes basan su funcionamiento en el procesado de secuencias de información, agregando al cómputo de cada elemento de la secuencia, el resultado del análisis del elemento anterior. Es decir, el resultado final de una RNN será una función que agregará el procesamiento de todos los elementos de la secuencia. Sin embargo, a medida que la red procesa más elementos de la cadena, tiene problemas en recordar información pasada.

Este problema se debe a las consecuencias del algoritmo mediante el cual las redes neuronales recurrentes son capaces de entrenar, el algoritmo de propagación hacia atrás en el tiempo o *back-propagation throug time* [54]. A medida de que la RNN recibe elementos de la secuencia y por tanto se desarrolla a lo largo de los time steps, el gradiente del error se propaga hacia atrás del mismo modo que lo haría en una red normal. Y del mismo modo que sucede en las redes habituales, a medida que dicho gradiente alcanza las capas más cercanas al input (en el caso de las RNN, a los primeros time steps procesados), decae exponencialmente. Es por esto que las RNN más sencillas no son capaces de aprender patrones muy extendidos en el tiempo, si no que solo son eficaces en rangos cortos, como por ejemplo secuencias de 10 elementos.

Para mitigar el problema de *short-term memory*, aparece un nuevo tipo de celda de memoria que sí es capaz de extraer patrones de secuencias de mayor longitud. Esta celda más compleja se conoce como *Long-Short Term Memory* o LSTM.

LSTM

Las celdas recurrentes conocidas como *Long Short-Term Memory* o LSTM fueron propuestas por Hochreiter y Schmidhuber en 1997. [19]. El objetivo de esta propuesta es precisamente resolver el problema por el cual las redes neuronales recurrentes no son capaces de recordar información de time step pasados en secuencias con un alto número de steps. Las celdas LSTM se comportan de manera natural aprendiendo secuencias que involucran largos periodos temporales, sin sufrir problemas de retardo en la convergencia.

La estructura dentro de una celda básica de una RNN se compone simplemente por una función de activación tipo *tanh*. Sin embargo, las celdas tipo LSTM tienen una estructura interna más compleja. Mientras que las RNN simples solo mantienen un vector de estado $h_{(t)}$, las celdas LSTM dividen el estado interno en dos vectores $h_{(t)}$ y $c_{(t)}$, pudiéndose interpretar el primero como *short-term* y el segundo como *long-term*.

Como puede verse en la Figura 2.15, el vector *long-term* $c_{(t)}$, atraviesa la celda de izquierda a derecha. La idea fundamental es que la red sea capaz de aprender cómo modelar la información de este vector. Para eso, puede eliminar o añadir información por medio de las puertas o *gates*. Las puertas son estructuras formadas por una función *sigmoide*, que decide la cantidad de información que seguirá el flujo y una operación producto. Para comprender mejor cómo funciona el flujo de información, se puede explicar qué sucede desde la entrada de datos hasta la salida.

La información entra a la celda a través del vector $x_{(t)}$, y el estado anterior a través de los vectores $c_{(t-1)}$ y $h_{(t-1)}$. Dentro de la celda, el procesamiento se lleva a cabo por cuatro capas densas con funciones de activación: una *tanh* y dos *sigmoide* (representadas por romboides en la imagen).

En primer lugar, la estructura interna se encarga de decidir qué información retirar del estado *long-term* $c_{(t)}$. Para ello entra en juego la primera de las capas *sigmoide*, la cual toma como entrada $h_{(t-1)}$ y $x_{(t)}$ y devuelve valores entre cero y uno (función $f(t)$ en la imagen). Esta salida actúa finalmente con el flujo de $c_{(t)}$ mediante la puerta *forget gate*. El calculo de $f(t)$ se muestra en la Ecuación 2.3.

$$f(t) = \sigma(W_{xf}^T x_{(t)} + W_{hf}^T h_{(t-1)} + b_f) \tag{2.3}$$

A continuación, se decide qué información ha de ser añadida al vector *long-term* $c_{(t)}$.

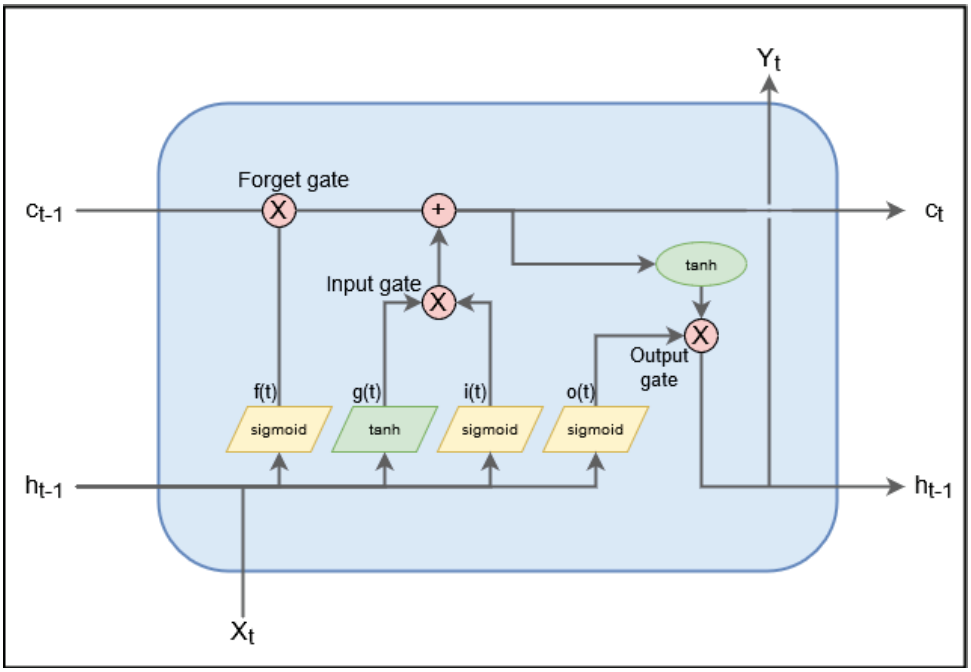

Figura 2.15: Estructura de una celda LSTM.

En esta operación entran en juego dos capas más. En primer lugar tenemos la capa densa con función de activación *tanh*. Esta capa es la única presente en las ya vistas RNN, por lo que su funcionamiento es el mismo, dando como salida $g(t)$ calculado en la Ecuación 2.4. En segundo lugar, también interviene una capa densa con función de activación *sigmoide*, cuya salida $i(t)$ controlará qué cantidad de $g(t)$ se añadirá al vector *long-term* $c_{(t)}$. Si $i(t)$ devuelve 0, la puerta *input gate* estará totalmente cerrada y si devuelve 1 estará totalmente abierta. El cálculo de $i(t)$ se muestra en la Ecuación 2.5.

$$g(t) = tanh(W_{xg}^T x_{(t)} + W_{hg}^T h_{(t-1)} + b_g) \tag{2.4}$$

$$i(t) = \sigma(W_{xi}^T x_{(t)} + W_{hi}^T h_{(t-1)} + b_i) \tag{2.5}$$

En último lugar, se decide cuál va a ser la salida de la celda LSTM filtrando el valor de $c_{(t)}$. Para ello se involucra la última capa densa con función de activación *sigmoide*, que decide qué cantidad de información de los vectores $h_{(t-1)}$ y $x_{(t)}$ se devolverá. Puede verse el cálculo en la Ecuación 2.6. Después de haber efectuado las operaciones de adición y sustracción de información sobre el vector $c_{(t)}$, este se procesa en una capa *tanh* y se

multiplica por el valor $o(t)$ en la puerta *output gate*.

$$o(t) = \sigma(W_{xo}^T x_{(t)} + W_{ho}^T h_{(t-1)} + b_o) \tag{2.6}$$

Este tipo de redes neuronales recurrentes son las más habituales en la literatura referente a la composición musical automática, por lo que conviene detenerse a revisar algunas de las propuestas que se han hecho.

En 2002, D. Eck *et al.* presentaron su investigación sobre composición musical utilizando LSTM [10]. Su objetivo era superar las limitaciones de las RNN para presentar estructura global y componer piezas más complejas. Para sus experimentos usaron un dataset de música blues típica de 12 compases. En su primer experimento, entrenaron el modelo únicamente con acordes, de modo que fuese capaz de generar cierta estructura armónica con ausencia de una línea melódica. Los resultados fueron satisfactorios, viendo como la red una vez aprendía a generar un ciclo de acordes, podía repetir dicho ciclo sin necesidad de más entrenamiento y por tanto inversión de tiempo y recursos. El segundo experimento consistía en generar melodía y acordes. De igual modo, la red fue capaz de aprender estructura y melodía (contenida en dicha estructura).

Otro ejemplo destacable es el modelo propuesto en 2016 por Sturm *et al.* para la generación de música celta [50]. La arquitectura contaba con tres capas ocultas con 512 celdas LSTM en cada una. El tipo de representación usada para la información musical fue la representación textual en formato ABC (ver Sección 2.2.3). En la etapa generativa, la salida de la red (siguiente nota a tocar) viene determinada por una distribución probabilística sobre el vocabulario (número único de tokens). La música generada efectivamente pertenece al genero musical del dataset de entrenamiento.

Otro caso en el que se pueden encontrar este tipo de celdas es en la arquitectura presentada en 2015 por Daniel Johnson en su blog personal Hexahedria [24]. En este artículo, Daniel propone lo que el llama *biaxial RNN*, una combinación de dos bloques recurrentes LSTM. El primero de ellos, recurrente a lo largo del tiempo, atiende a la relación temporal entre las notas de la secuencia, descubriendo patrones entre ellas. El segundo bloque es recurrente a lo largo del eje de las notas, teniendo entradas desde las notas más graves hasta las más agudas y recogiendo patrones de notas tocadas de manera simultánea. Es decir, no solo se presenta recurrencia en la dimensión temporal, como se ha visto hasta ahora, si no también en la dimensión tonal. Los datos fueron extraídos de un dataset de música de piano clásica en formato MIDI [29] y adaptados

en formato piano roll. En dicha representación cabe destacar el añadido de información extra para representar el contexto de cada elemento analizado en la secuencia (notas previas, posición en el compás, etc.).

Así por tanto, se constituyen las LSTM como celdas más complejas que las RNN típicas. Son capaces de procesar secuencias de mayor longitud temporal, trabajando con la información de modo que sea capaz de preservarla o desecharla según se requiera. Sin embargo, las celdas LSTM no son la única alternativa a las RNN clásicas, si no que existe todo un conjunto de variantes. Una de las más famosas son las celdas tipo GRU (*Gated Recurrent Unit*).

GRU

Este tipo de celda fue propuesta en 2014 por Kyunghyun et al. [5]. Es una versión simplificada de la celda LSTM, pero se ha demostrado que pueden trabajar con el mismo rendimiento en general [42] [16].Como se ve en la Figura 2.16, en este caso la celda solo trabaja con un único vector de estado $h_{(t)}$ y con dos puertas que modificarán el flujo de dicho vector. La primera de ellas, *update gate*, cuya salida es $z(t)$ sirve como controladora de las puertas *input gate* y *forget gate* de modo que su rango de valores de salida [0 - 1] controla si se elimina información del estado, o se añade. Su cálculo puede verse en la Ecuación 2.7.

$$z(t) = \sigma(W_{xz}^T x_{(t)} + W_{hz}^T h_{(t-1)} + b_z) \tag{2.7}$$

La segunda, *reset gate*, cuya salida es $r(t)$, controla qué parte del estado anterior se le mostrará a la capa principal con función *tanh* $g(t)$, que será la que de lugar a la salida de la celda. El calculo de de $r(t)$ puede verse en la Ecuación 2.8 y el de $g(t)$ en la Ecuación 2.9.

$$r(t) = \sigma(W_{xr}^T x_{(t)} + W_{hr}^T h_{(t-1)} + b_r) \tag{2.8}$$

$$g(t) = tanh(W_{xg}^T x_{(t)} + W_{hg}^T (r(t) \otimes h_{(t-1)}) + b_g) \tag{2.9}$$

Un ejemplo del uso de estas celdas en el dominio de la composición musical automática puede verse en el estudio realizado por Chung *et al.* [7]. En el, los autores comparaban

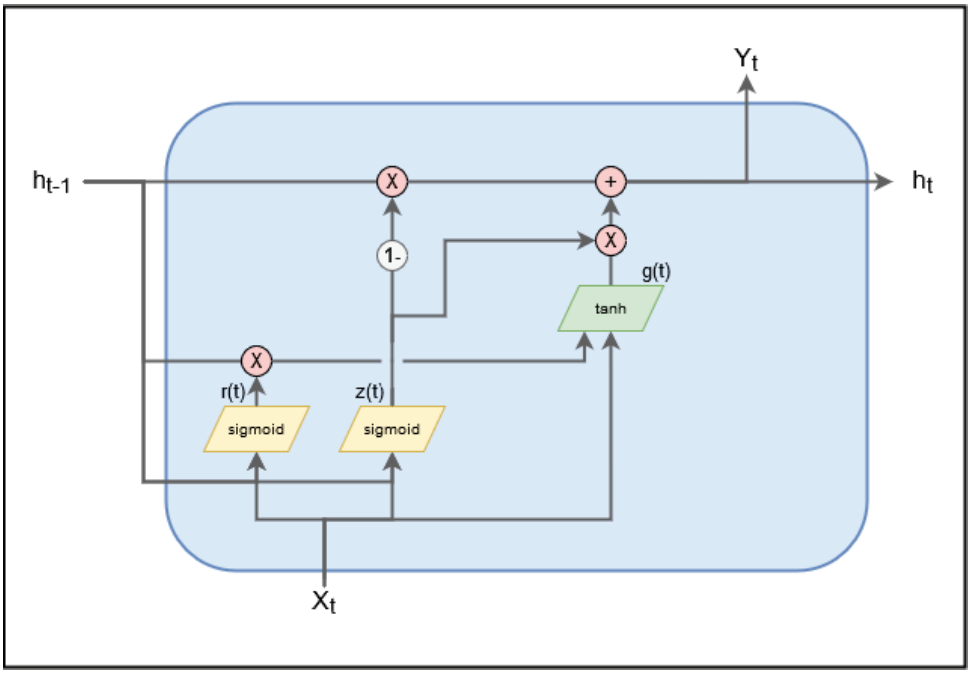

Figura 2.16: Estructura de una celda GRU.

los resultados de tres modelos de composición automática, cada uno generado con un tipo de celda distinto (RNN, LSTM y GRU). Para ello, trabajaron con tres dataset de musica polifónica en formato MIDI. Los autores concluyen que la arquitectura con celdas GRU superó en precisión al resto de modelos, aunque sin un gran distanciamiento respecto a las LSTM.

Otro ejemplo es el sistema *GRUV*, propuesto por Nayebi y Vitelli [39]. En esta ocasión, se trabajó con ondas de audio como datos de entrada. Los autores, compararon de nuevo la arquitectura GRU con la LSTM, resultando en este caso un mejor resultado para el modelo con celdas LSTM.

Redes neuronales recurrentes bidireccionales

En 1197, Schuster y Paliwal presentaron un modelo que derivaba de las tradicionales redes neuronales recurrentes: la redes neuronales recurrentes bidireccionales [47]. La motivación residía en hacer que la predicción de la red no dependiese únicamente de los datos vistos en steps anteriores, si no también en datos de steps futuros.

Como puede verse en la Figura 2.17, la idea estructural consiste en dividir las celdas recurrentes en dos, de manera que una de las topologías analice la secuencia de datos siguiendo una dirección temporal positiva, y la otra topología analice la secuencia de

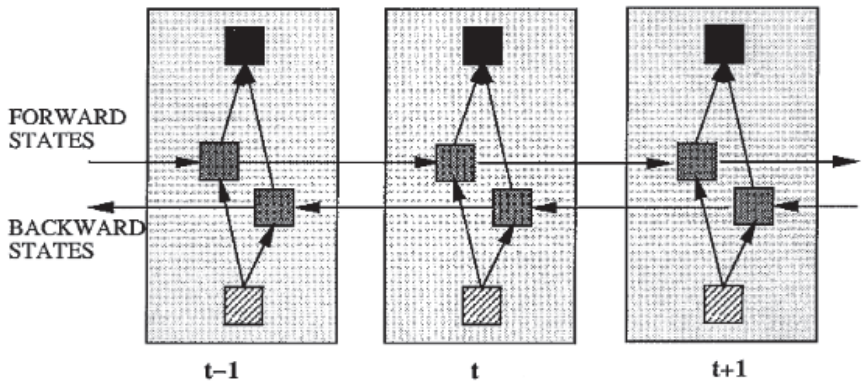

Figura 2.17: Arquitectura de una red neuronal recurrente bidireccional extraída del artículo de Schuster y Paliwal [47].

manera inversa. La salida de la red en cada time step será una combinación de ambas celdas.

En la literatura podemos encontrar ejemplos de este tipo de arquitectura aplicado al caso de la composición musical. Por ejemplo, en 2017, Lim *et al.* propusieron un sistema basado en LSTM bidireccionales (*BLSTM*) para generar acompañamiento musical en forma de una progresión de acordes dada una melodía [33]. La arquitectura consistía en una red bidireccional con dos capas LSTM de 128 celdas cada una. Esta red fue entrenada con un corpus de 2252 partituras de musica moderna (rock, jazz, pop, etc), resultando en un total de 1802 canciones. Para representar las notas de la melodía, se tuvieron en cuenta 12 posibles valores (no se contó con ninguna octava extra) y para los acordes, únicamente se usaron triadas. El entrenamiento se llevó a cabo presentando a la red varios grupos de cuatro tiempos (negras) de melodía y comparando a la salida los acordes correspondientes. En la fase de generación, iterativamente se pasó a la red cadenas de cuatro tiempos, concatenando los acordes de salida hasta tener la progresión completa. Los autores compararon el sistema con otros modelos y lo evaluaron cuantitativamente (matrices de confusión) y cualitativamente (con 25 oyentes). Los resultados mostraron mayor precisión que con los otros modelos, así como una tendencia por parte de los usuarios a elegir estas composiciones frente a las de los otros modelos.

Mecanismos de *Attention*

El mecanismo de *Attention* fue presentado en 2014 por Dzmitry Bahdanau *et al* como extensión para mejorar el rendimiento del modelo *Encoder-Decoder* (ver Sección 2.3.3)

en la tarea de la traducción automática [3].

En un *encoder-decoder* convencional, el *encoder* representa mediante un vector de tamaño fijo (estado oculto final) toda la información de la frase de entrada. En palabras de los propios autores, "*el rendimiento de un encoder-decoder básico decae rápidamente a medida que la longitud de la cadena de entrada crece.*". El mecanismo de *Attention* pretende solventar precisamente este problema. Mediante esta técnica, un conjunto de vectores es generado en lugar de un solo vector de dimensión fija. Entre dichos vectores se selecciona un subconjunto cercano al término que se pretende traducir en cada time step.

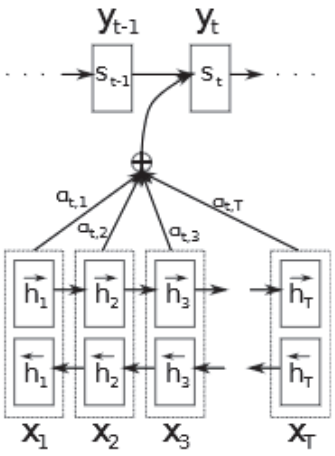

Figura 2.18: Arquitectura de attention extraída del articulo original [3].

En la Figura 2.18, puede verse un esquema de la arquitectura de un *encoder-decoder* intentando generar la t-ésima palabra objetivo y_t dada la frase (x_1, x_2, ..., x_T). Como se ve, en lugar de mantener como vector final el estado oculto de la celda en el ultimo step del *encoder*, se consideran todas las salidas intermedias hasta el momento y se computa la suma ponderada de dichos vectores intermedios. Este cálculo permite determinar qué palabras serán más relevantes para el *decoder* en ese step.

Existen varias propuestas basadas en este mecanismo:

- ***Visual attention***: Fue propuesto en el año 2015 por Xu *et al.* [55]. El objetivo de esta propuesta consistía en un sistema capaz de alinear la imagen de entrada y producir una palabra que la describiese como salida. La arquitectura consistía en una red convolucional que extraía características de la imagen y a continuación una red recurrente con mecanismos de *attention* para dar como salida la palabra. El mecanismo de *attention* consigue focalizarse en ciertos elementos de la imagen que la definen de manera que la frase de salida se corresponda con dichos elementos.

- ***Herarchical attention***: Esta propuesta llego en 2016 de la mano de Yang *et al.* [57]. Se trata de un sistema de clasificación de documentos que aplica los mecanismos de *attention* en dos niveles. La arquitectura presenta dos módulos *encoder-decoder + attention*. El primero de ellos aplicado a nivel de frase y el segundo a nivel de palabra.

- ***Transformer***: Como se comentó anteriormente, la consecuencia más interesante de la aparición de los mecanismos de *attention* fue la aparición de los modelos *Transformer*, propuestos por Vaswani *et al.* en 2017 [52]. Esta arquitectura supuso un nuevo nivel de mejora en el campo del Procesamiento del Lenguaje Natural. Este tipo de sistema permite alinear ciertas palabras de una secuencia con otras, calculando una representación de dicha secuencia mucho más precisa y eficiente que en anteriores modelos.

Por último también se pueden encontrar ejemplos de estos sistemas en el campo de la generación musical por computador. Por ejemplo, en 2019 Guan *et al.* presentaron un modelo basado en redes GAN (ver Sección 2.3.2) añadiendo mecanismos *attention* para la generación de musica multi-instrumento. Esta implementación conseguía extraer más características en la dimensión temporal.

Otro ejemplo es el modelo *Music Transformer*, propuesto pro Huang *et al.* en 2018 [22]. El sistema propuesto consigue generar largas secuencias musicales mediante la modificación del algoritmo *attention* en el *transformer*. Esta modificación permite reducir la cantidad de memoria necesaria para representar la cadena, permitiendo así cadenas de mayor longitud.

2.3.4 Métodos de evaluación

La característica diferenciadora de los modelos de composición automática es que el humano no tiene interacción de ningún tipo con el sistema más allá de la previa elección del conjunto de datos de entrenamiento y la configuración de los parámetros para conseguir un buen resultado. Es decir, el propio algoritmo devuelve una composición musical propia, sin intervención humana. Esto supone un problema a la hora de evaluar dichos modelos, ya que al contrario que pasa en por ejemplo, problemas de clasificación, la música generada no es algo que pueda medirse numéricamente para calcular el error. Es por eso por lo que después de obtener la pieza musical, es necesario seguir un proceso de validación externo en el que sí intervenga un humano.

El método más inmediato en el que se puede pensar para llevar a cabo una valoración de algo tan subjetivo como la música, es el Test de Turing. Existen numerosas aportaciones que utilizan el Test de Turing como [1] [2]. Como cabría esperar, el procedimiento de este sistema de evaluación consiste en presentar obras generada artificialmente junto con otras obras creadas por humanos a varios oyentes. Estos deberán decidir si se trata de una obra generada por computador o una obra compuesta por un humano. El problema de estos estudios es que entran en juego numerosos factores como la calidad del audio, el entorno donde se desarrollen estos test, etc. Estos problemas, unidos a los diferentes niveles de comprensión musical de los sujetos pueden no resultar en una métrica aceptable [56].

Sin embargo, no todos los métodos de evaluación están basados en la intervención humana en el sentido de tomar parte del proceso de juicio. Existen estudios que desarrollan métricas más empíricas que permiten evaluar de manera objetiva. Yang en su trabajo [56] divide estas métricas en tres tipos:

- En el primer grupo se encuentran las **medidas probabilísticas sin conocimiento de dominio musical**. Inicialmente concebidas para tareas de generación de imagen, también aparecen relacionadas con creación musical autónoma. Estas métricas se corresponden con cálculos de errores típicos de modelos estadísticos, como por ejemplo *log-likelihood*.

- En segundo lugar, Yang habla de las **métricas especificas de la tarea/modelo** como respuesta a la gran variabilidad que existe entre los diferentes métodos de generación musical. Como señala el autor, el problema de este tipo de métricas es precisamente ese, la gran variabilidad que existe. Esto conduce a dificultades a la hora de realizar comparativas entre ellas así como a obtener una evaluación que no se vea sesgada.

- El último grupo es el formado por las **métricas basadas en conocimiento general del dominio musical**. Estas métricas son propias del dominio en el que se enmarcan. Es decir, en este caso no se usarán métricas estadísticas clásicas ni tampoco métricas desarrolladas para probar la efectividad de un modelo en concreto. Se genera así un conjunto de métricas aplicables a la mayoría de los sistemas de generación automática musical, puesto que evalúan aspectos intrínsecos a la música como el tono, la frecuencia, etc. Por ejemplo, Yang señala el trabajo de Chuan et al., quienes utilizan métricas de tono y frecuencia que explican la eficiencia del mode-

lo comparando los diferentes métodos de extracción de características utilizados [6].

En este sentido, Yang propone por tanto un método utilizable como métrica para comparar la música generada automáticamente con la música usada como entrenamiento y que está basado en conocimiento del dominio musical [56].

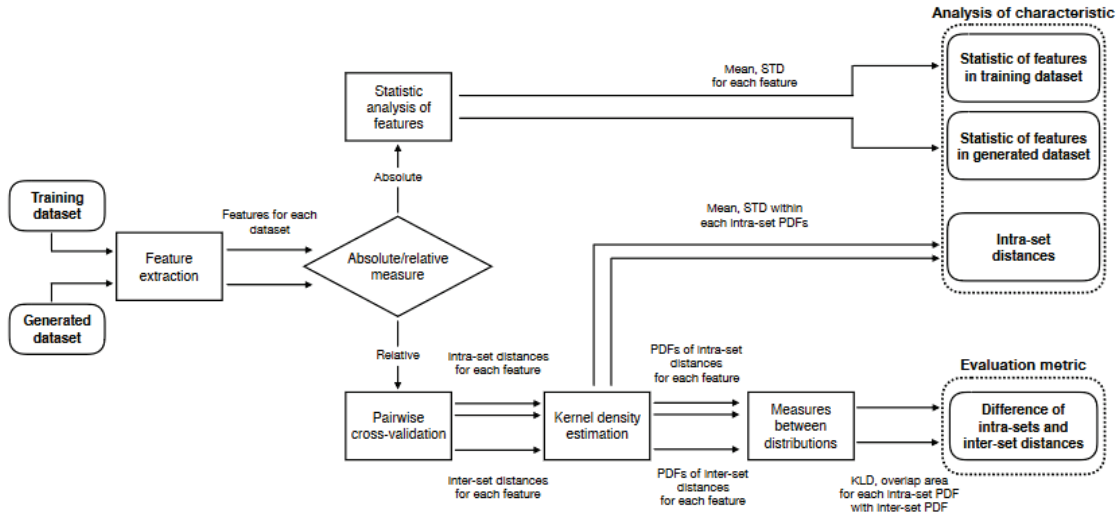

Fig. 1: General work flow of the proposed method

Figura 2.19: Sistema de evaluación propuesto por Yang [56].

Como puede verse en la Figura 2.19, el método propuesto se basa en el siguiente flujo de trabajo. En primer lugar el sistema toma pistas de audio en formato MIDI correspondientes al resultado generado por el modelo así como al dataset de entrenamiento. A continuación se extraen un total de 9 características de dichos audios tales como el tono, el numero de notas, intervalo entre notas, ect. Con estos datos, se realizan dos medidas distintas: en primer lugar se realiza una medida absoluta para extraer datos sobre el modelo generador (rama superior de la figura) y en segundo lugar una medida relativa (rama inferior de la figura) comparando ambos audios en varios de los aspectos extraídos anteriormente. Tras estos cálculos, se generan dos métricas que evalúan las distancias entre ambas muestras así como la eficiencia del generador.

3 Desarrollo

Hasta el momento se ha realizado una revisión del estado del arte en el dominio de la síntesis musical mediante computador y más específicamente de la composición musical automática mediante técnicas de redes neuronales profundas o *Deep Learning*. En concreto, se ha mostrado como ciertos modelos de *Deep Learning* son adecuados para acometer esta tarea, destacando entre todos ellos las arquitecturas de redes neuronales recurrentes.

La segunda parte de esta tesis consiste en la implementación de algunos de estos modelos con el fin de comprender cómo se diseña una arquitectura de este tipo. El objetivo es también comprobar las capacidades que tienen las herramientas revisadas para generar música de manera que resulte agradable al el oyente. En concreto se pondrán a prueba 3 modelos para generación polifónica de una pista de piano.

En primer lugar se presentará el proceso seguido para la selección y preparación de los datos, explicando el método de representación implementado. En segundo lugar, se detallarán los diferentes modelos que serán probados, explicando el proceso de entrenamiento así como el proceso de generación. Por último, se expondrán los experimentos que se han llevado a cabo, analizando los resultados obtenidos para cada uno de ellos.

3.1 Preparación de datos

El primer punto a tener en cuenta antes de diseñar e implementar las arquitecturas es definir cómo se van a tratar y representar los datos con los que se trabaje.

Respecto al formato, se ha optado por buscar datos representados simbólicamente. Más concretamente, por su alta disponibilidad en Internet, se buscarán piezas en formato MIDI (ver Sección 2.2.3).

Una vez obtenidos los datos para entrenar la red, el siguiente paso es procesar la in-

formación de manera que sea interpretable por los modelos de *Deep Learning*. Para ello, se hará uso del lenguaje de programación Python y más concretamente de la biblioteca *music21* [8]. Este paquete de Python implementa un conjunto de herramientas que permite trabajar de manera cómoda y sencilla con archivos MIDI. Desde trabajar con las pistas individualmente, hasta generar archivos de audio, partituras o otras pistas MIDI.

3.1.1 Representación de los datos

En este momento, es crucial seleccionar el tipo de representación con la que se quiere trabajar. Como se explicó en la Sección 2.2.3, en este caso se van a representar los datos en formato textual, en notación americana. Esta representación ha sido extraída del blog de Sigurður [49]. Como se ve, la representación propuesta cubre el caso de las notas y los acordes. Sin embargo para reducir la complejidad, establece una duración de nota constante, lo cual resta riqueza a la capacidad de generación del sistema. Con el objetivo de capturar la mayor información posible, se ampliará la representación, de modo que se explicite la duración de las notas y acordes. De este modo, estos serán los elementos usados:

- ***Notas***: Como se ha mencionado, las notas se representarán mediante su correspondiente letra en notación americana así como su alteración si esta existe. Además, se añade numéricamente la octava de la nota. Por ejemplo, un sol sostenido en su cuarta octava será representado así: "*G#4*". Los silencios vendrán dados por el término *rest*.
- ***Acordes***: Los acordes, intervalos y todo tipo de notas que se reproduzcan al mismo tiempo se representarán como una sucesión de notas separadas por el carácter punto. De esta forma el acorde de do menor se representará así: "*C4.E-4.G4*".
- ***Duración***: La duración se representa de manera explícita. Con esta estrategia se consigue plasmar la noción de mantener una nota pulsada además de permitir al sistema generar cierta estructura rítmica. El valor de la duración se añade a continuación de las notas y acordes separado por el carácter "_". De esta manera, la nota de sol sostenido en su cuarta octava con duración de una negra será: "*G#4_1.0*".

Una vez extraídos y preprocesados los datos, es el momento de comenzar el diseño e implementación de las arquitecturas.

3.2 Modelo RNN vector-sequence (stateless)

La primera arquitectura probada es tomada de la entrada de blog de Sigurður Skúli [49], en la cual realiza una introducción a la composición automática con RNN mediante la implementación de una arquitectura con LSTM. A continuación se explicará este modelo, desde la gestión de los datos para ser introducidos en la red, hasta la arquitectura de la red que se probará.

3.2.1 Arquitectura

La arquitectura propuesta tiene ciertas características que se deben comentar. En primer lugar, se trata de una topología *stateless*. Esto quiere decir que el estado oculto h_t de las celdas de memoria no se guarda de cara al siguiente *batch* de entrenamiento, sino que se reinicia. Es decir, la red aprende con cadenas aleatorias en cada time step, sin correlación entre ellas. Esto permite mayor flexibilidad con los datos de entrada, ya que no es necesario mantener correlación entre los *mini-batches* para que la red se entrene adecuadamente (esto se verá en la explicación del siguiente modelo).

Además el modelo se diseña como un *sequence-vector* (ver Sección 2.3.3). Es decir, recibe como entrada una secuencia de datos y no es hasta el último time step de la secuencia cuando el modelo devuelve su predicción. A continuación, en la Figura 3.1 puede verse la arquitectura implementada desarrollada a lo largo del tiempo.

En primer lugar se dispone una capa recurrente con 512 celdas LSTM (ver Sección 2.3.3) de tipo *sequece-sequence*. Recibe una secuencia de entrada y a cada time step devuelve el resultado a la capa posterior. Además esta capa cuenta con un *recurrent dropout* con valor 0'3. La siguiente capa es exactamente igual a la primera.

La tercera capa, también formada por 512 celdas LSTM, es de tipo *sequence-vector*. No será hasta el último paso de procesamiento cuando devuelva el resultado final. El conjunto de capas recurrente, por tanto, puede ser visto como *sequence-vector*.

A continuación se aplica la operación de normalización del *batch* y se dispone una capa densa con 256 unidades, parámetro de *dropout* igual a 0'3 y función de activación *ReLu*.

Por último, se dispone una capa densa con número de neuronas igual al total del vocabulario con el que se trabaje en el experimento. La función de activación será una

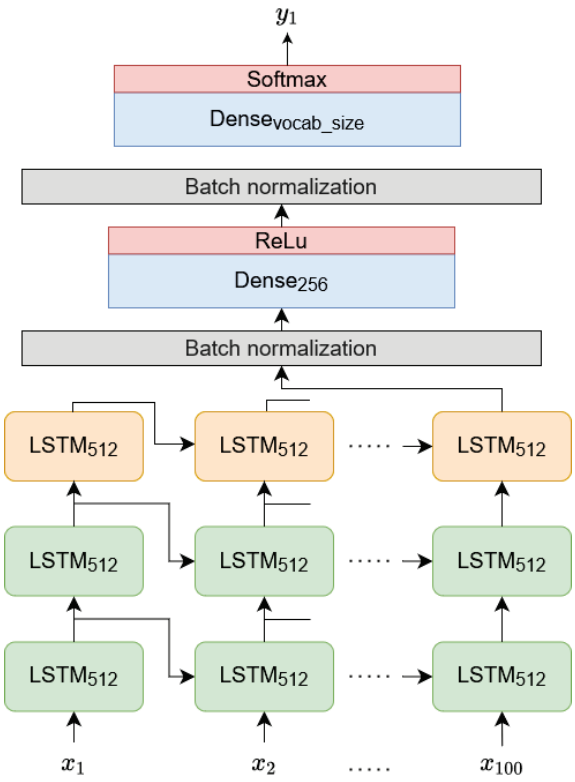

Figura 3.1: Arquitectura del modelo sequence-vector (sequence-vector).

Softmax, de modo que el comportamiento de toda la arquitectura será el de un modelo de clasificación. Es decir, el sistema recibe una serie de notas y acordes a lo largo del tiempo y deberá clasificar la salida en una de las notas o acordes del vocabulario conocido.

Cabe señalar que en la imagen solamente se están procesado 100 time steps, correspondientes a los 100 elementos de uno de los ejemplos de entrenamiento. En el siguiente apartado se detallarán los datos de entrada.

3.2.2 Fase de entrenamiento

Una vez diseñada la arquitectura, es el momento de comenzar el entrenamiento de la red. Para ello, se deben gestionar los datos de entrenamiento y adecuar su estructura para ser consumidos por la red. Se parte por tanto del resultado obtenido con el proceso seguido en la Sección 3.1.1.

Para el proceso de codificación de los datos se va a emplear la estrategia de valores numéricos discretos (ver Sección 2.2.4). Cada uno de los diferentes tokens del conjunto

de datos será mapeado a un entero diferente. El conjunto de valores será normalizado previamente a la ingesta por parte del modelo. Cabe destacar que el tamaño del vocabulario variará en función del número de canciones que se tomen del total del dataset. En el siguiente capítulo se comentarán los experimentos y se indicará el número de canciones tomadas para entrenamiento..

Como se ha explicado, el modelo es tipo *sequence-vector*. Atendiendo a esto, los datos de entrada serán por tanto vectores con cierta longitud, determinada por el parámetro *sequence length* y los datos de *target* serán el elemento posterior a dicha secuencia.

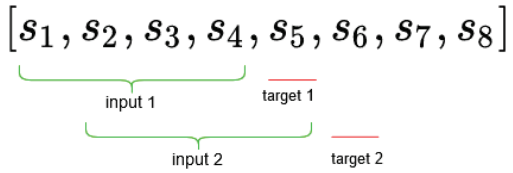

Figura 3.2: Ejemplo datos de entrenamiento con longitud de secuencia igual a 4. En verde la secuencia de entrada y en rojo su correspondiente valor target.

La Figura 3.2 muestra un ejemplo de estructuración de datos para el proceso de entrenamiento con el parámetro de longitud de secuencia igual a cuatro. Como se ve, se toman los n primeros elementos del total de datos disponibles y es el $n+1$ el que se dispone como valor de salida deseado. La siguiente secuencia de entrenamiento se construye desplazando todo un elemento a la derecha. En el caso de este modelo, se ha seleccionado un valor de longitud de secuencia igual a cien elementos. El total de ejemplos se empaquetarán en *mini-batches* de procesamiento. El tamaño del *batch* se detallará en los experimentos.

Por tanto, las dimensiones de la entrada a la red para cada *batch* serán 3: (tamaño *batch*, longitud secuencia, 1). Cada *batch* tendrá la siguiente forma:

$$[[x_1, ..., x_{100}], [x_2, ..., x_{101}], ...]$$

3.2.3 Fase de generación

Una vez la red ya ha entrenado y el error en la predicción es relativamente bajo, es el momento de comenzar el proceso de generación. Es en este punto donde destaca la capacidad creativa de este tipo de arquitecturas.

Como se ve en la Figura 3.3, la red recibirá una secuencia de la misma longitud que en

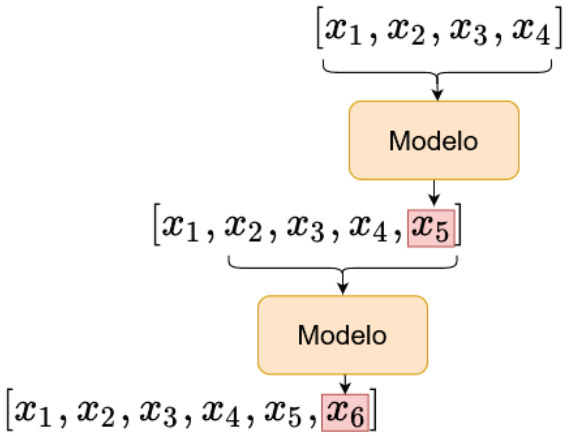

Figura 3.3: Ejemplo de generación con la arquitectura propuesta. Tamaño de secuencia igual a 4.

la fase de entrenamiento, es decir, de cien elementos y deberá generar un elemento extra que se añadirá a dicha secuencia. En el siguiente paso de generación, la cadena recibida por la red contendrá los últimos 100 elementos, incluyendo el generado y excluyendo el primero. Mediante esta aproximación, se generarán secuencias novedosas de longitud arbitraria en base al modelo entrenado.

3.3 Modelo RNN sequence-sequence (stateful)

El segundo modelo probado se toma del libro de Géron *Hands On Machine Learning* [12]. En uno de los capítulos del libro, se presenta a modo de ejemplo una arquitectura recurrente para resolver el problema de la generación automática de texto. Este modelo, va generando carácter a carácter hasta construir nuevas palabras y frases. La representación de datos seguida en esta tesis permite que el problema de la composición automática pueda modelarse como generación de caracteres. Es por esto que se considera este modelo como propuesta a evaluar.

3.3.1 Arquitectura

En este caso la arquitectura propuesta es de tipo *stateful*. Esto significa que tras cada step i durante el procesamiento de un ejemplo, el estado oculto de las celdas de memoria es almacenado. Este valor se usa como estado oculto inicial a la hora de procesar el step

i del siguiente *batch*. Dado que el entrenamiento de la red se realiza mediante largas secuencias sucesivas divididas, esto tiene ciertas implicaciones que deben tenerse en cuenta a la hora de preparar los datos.

Otro dato a tener en cuenta es que el modelo en este caso es tipo *sequence-sequence* (ver Sección 2.3.3. Por cada step o dato dentro de la secuencia de entrada, la red devolverá un resultado.

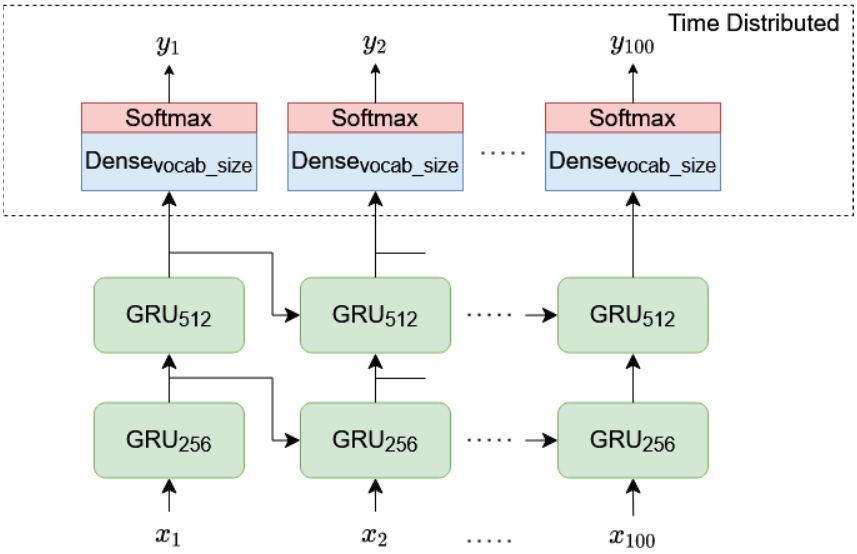

Figura 3.4: Arquitectura del modelo sequence-sequence (stateful).

En la Figura 3.4 puede verse la arquitectura correspondiente. En este caso, se disponen dos capas recurrentes con celdas tipo GRU (ver Sección 2.3.3) ambas con configuración *sequence-sequence*. De igual modo, en ambas capas se configura un *recurrent dropout* de 0'3 con el objetivo de reducir el *overfitting*.

La salida de esta primera fase recurrente, es introducida en una capa densa. La principal característica de esta capa densa es que es de tipo *time distributed*. Esto significa que la misma capa densa es aplicada al resultado arrojado por la segunda capa recurrente cada time step. De nuevo, el número de neuronas será igual al tamaño del vocabulario y la función de activación de dicha capa densa será la función *Softmax*.

En la imagen solo se muestra el procesamiento de un *batch*. En este caso, los *batch* solamente pueden contener una secuencia input y una *target*, se explicará en el siguiente apartado.

3.3.2 Fase de entrenamiento

A continuación, se deben trabajar los datos para entrenar la arquitectura descrita. Se toman los datos tratados según se indica en la Sección 3.1.1 y se procede a su codificación y estructuración.

Para el proceso de codificación de los datos se va a emplear la estrategia de *one-hot encoding* (ver Sección 2.2.4). Cada uno de los diferentes tokens del conjunto de datos será representado mediante un vector binario. El número de canciones a transformar dependerá del experimento, por lo que se comentará en el capítulo siguiente.

El modelo es del tipo *sequence-sequence*, lo que implica que los datos de entrada serán una secuencia y los *target* serán la misma secuencia desplazada en un elemento a la derecha.

Otra característica a tener en cuenta es que el modelo es *stateful*, esto implica que los datos deben ser consecutivos de modo que el estado oculto de las celdas recurrentes pueda ser usado almacenado.

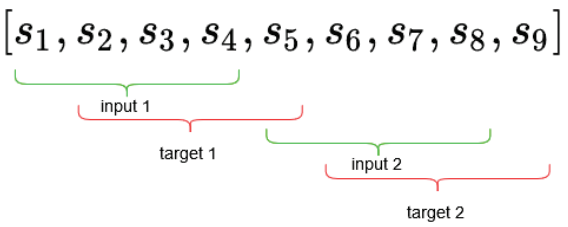

Figura 3.5: Ejemplo datos de entrenamiento con longitud de secuencia igual a 4. En verde la secuencia de entrada y en rojo su correspondiente secuencia target.

En la Figura 3.5 puede verse un ejemplo de secuencia de entrada y *target*. Cabe destacar que en este caso, el tamaño del *batch* deberá ser igual a uno, para que cuando se termine de procesar un *batch*, el siguiente contenga datos contiguos al anterior. Por ello para generar los ejemplos de entrada, se desplaza un total de unidades igual al tamaño de la secuencia. En este caso, dado que el tamaño del *batch* es uno, no hará falta su división en *mini-batches*.

En la imagen también puede verse una longitud de secuencia igual a cuatro. En el modelo implementado se selecciona un tamaño de cien elementos.

Resumiendo, las dimensiones de la entrada a la red serán por tanto: (tamaño batch, longitud secuencia, 1). Cada *batch* tendrá la siguiente forma:

$$[[x_1, ..., x_{100}]]$$

3.3.3 Fase de generación

En este momento, la red ha aprendido de los datos de entrenamiento y está preparada para comenzar a generar nuevas secuencias de notas y acordes. En este caso, el modelo no recibirá una secuencia de 100 elementos, si no que recibirá una nota o acorde inicial y deberá continuar dicha secuencia.

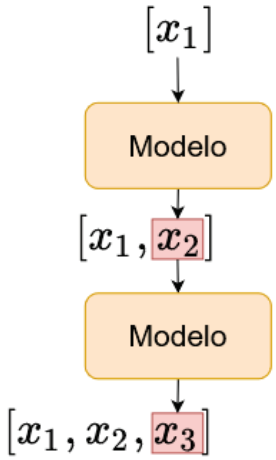

Figura 3.6: Ejemplo de generación con la arquitectura propuesta.

Como se ve en la Figura 3.6, el sistema recibe una nota u acorde inicial y da como salida otra nota o acorde. Será esta salida la que se use como entrada en el siguiente paso en el proceso de cogeneración. Este bucle puede repetirse un número arbitrario de veces, de modo que se genere una secuencia del tamaño deseado.

3.4 Modelo RNN Bidireccional + Attention (stateless)

El último modelo que se implementa es propuesto por Alex Issa en su blog [23]. A diferencia de los anteriores, esta arquitectura se caracteriza por implementar una red neuronal recurrente bidireccional (ver Sección 2.3.3) e incluir una capa de *attention* (ver Sección 2.3.3). El objetivo de implementar esta arquitectura es comprobar las capacidades de estas dos tecnologías a la hora de generar piezas musicales creativas.

La preparación de los datos para la fase de entrenamiento así como la de generación es igual a la expuesta en la Sección 3.2. Por ello, en este caso solo se explicará la arquitectura.

3.4.1 Arquitectura

La arquitectura descrita por Alex es de tipo *stateless*, es decir, las celdas recurrentes no almacenan su estado interno entre el procesamiento de cada *mini batch*. Además, se trata de una arquitectura *sequence-vector*. Puede verse en la Figura 3.7.

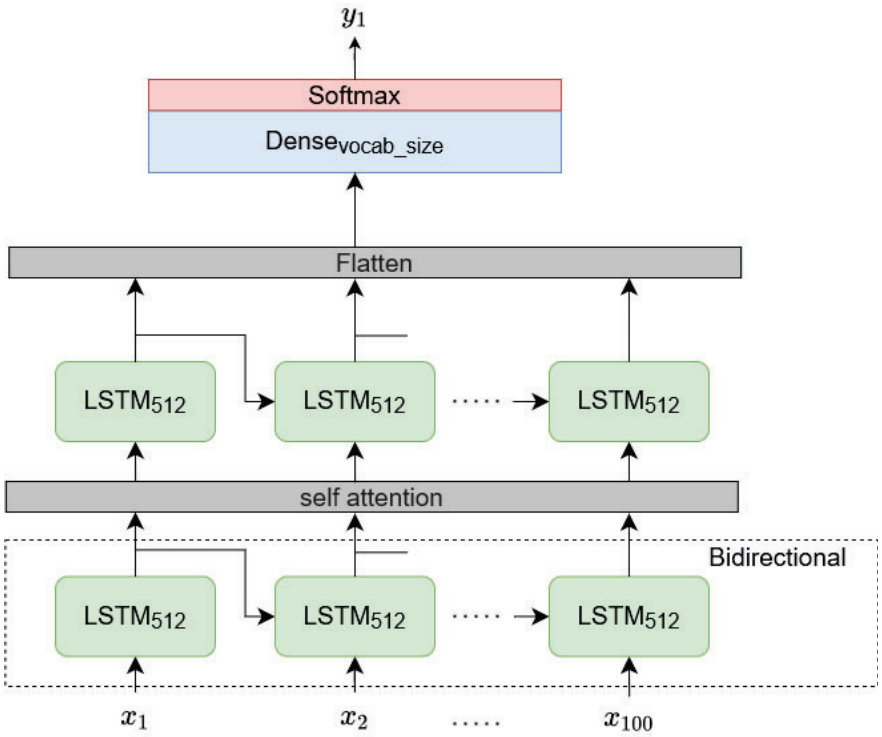

Figura 3.7: Arquitectura combinando RNN bidireccional y *attention*.

La principal característica de esta topología es el uso de un RNN bidireccional y una capa de *attention*. Como se ve en la imagen, en primer lugar se cuenta con una capa LSTM bidireccional de 512 celdas con configuración *sequence-sequence*. Esta capa cuenta con un parámetro de *dropout* con valor 0'3. La salida de esta capa es la entrada de una capa *self-attention* cuya principal función es extraer los principales valores de interés relacionados con el elemento procesado en ese momento. La salida de esta capa entra en otra capa recurrente con 512 unidades LSTM igual que la primera, tipo *sequence-sequence* y con parámetro *dropout* de 0'3, aunque en este caso sin ser bidireccional.

Antes de entrar en la capa densa, los valores devueltos son procesados por una capa *Flatten* cuya función es reducir su dimensión a uno. Esto hace que el resultado final de la arquitectura no sea una secuencia pese a no tener una capa recurrente final del tipo *sequence-vector*. Este vector unidimensional constituye la entrada para la capa densa, la cual está formada por tantas neuronas como elementos distintos haya en el vocabulario. La salida de esta capa es, de nuevo, la función *Softmax*.

Tras las pruebas experimentales insatisfactorias de esta arquitectura, se propone una modificación. Esta topología alternativa puede verse en la Figura 3.8.

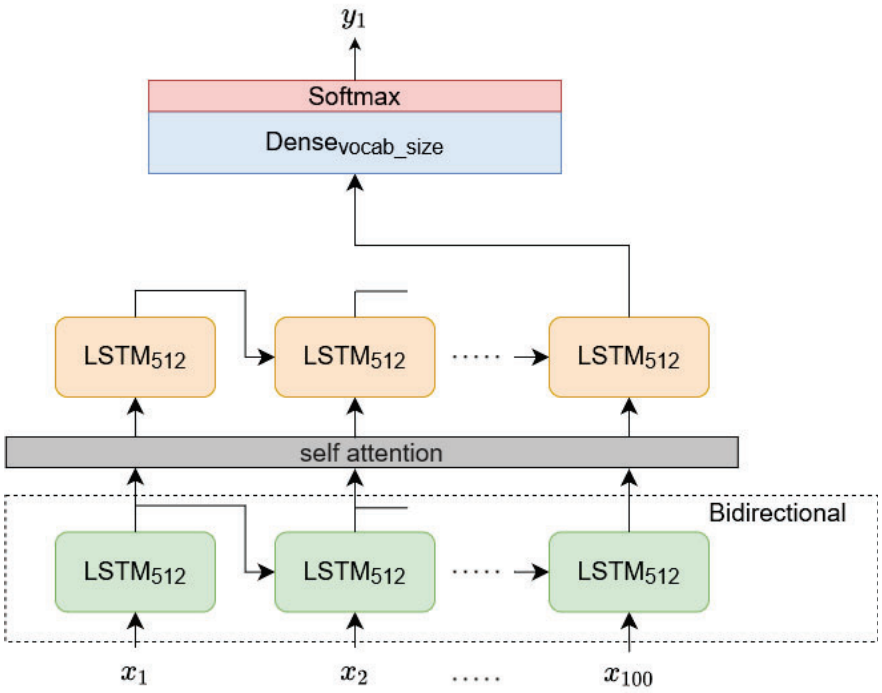

Figura 3.8: Variación de arquitectura combinando RNN bidireccional y *attention*.

En este caso se ha configurado una topología similar a la inicial, diferenciándose en que la segunda capa recurrente es de tipo *sequence-vector*. Por ello, la capa *Flatten* ha sido eliminada.

4 Experimentos

Una vez definidos los modelos es momento de probarlos experimentalmente. El primer paso para ello es seleccionar los datos que se usarán en el proceso.

Para llevar a cabo los experimentos, se ha determinado que lo más apropiado es emplear ejemplos de música clásica, dado que es el género más usado en la literatura. Se han buscado diversos datasets MIDI de música clásica que consistiese en una única voz de piano tocando melodía y acordes. De entre todos los datasets encontrados, se han seleccionado dos: *Classical Piano Midi Page* [29] y *midiworld* [36]. De dichas fuentes, se han extraído un total de 300 obras. El tratamiento de dichos datos se explica en la Sección 3.1.

A continuación se verán las diferentes pruebas realizadas sobre los modelos descritos con anterioridad. Estos experimentos han sido llevados a cabo en el entorno de Google *Collaboratory*, el cual pone a disposición de los desarrolladores entornos virtuales de ejecución con GPU. Además se ha hecho uso de Python y la biblioteca Tensorflow (y Keras), uno de los módulos más usados para trabajar con redes neuronales. Todos los ejemplos mencionados se podrán escuchar en el siguiente enlace: `https://soundcloud.com/juan-julian-cea-moran/sets`

Cabe destacar que el método con el que se evaluarán los ejemplo será puramente humano. Las piezas serán evaluadas por un músico y por varias personas sin conocimientos musicales profundos.

4.1 Modelo sequence-vector (stateless)

El primer modelo a probar es el modelo *sequence-vector* visto en la sección 3.2.

En primer lugar, se han realizado pruebas con la arquitectura base, usando celdas LSTM en las capas recurrentes. En estos experimentos se ha usado una longitud de se-

cuencia igual a 100 y 70 elementos, y un tamaño de *batch* de 128 y 64. De igual modo se han realizado pruebas tomando 100, 50 y 10 canciones del dataset (debido a que el procesamiento del dataset completo agotaba los recursos ofrecidos por la plataforma).

En dichas pruebas se concluyó que tanto en el caso en el que se usaban 100 canciones como en el que se usaban 50, la red era incapaz de entrenar adecuadamente, manteniendo el valor del error en rangos superiores a 7 % pese a cambios en los parámetros. Además, en ambos casos el tiempo de ejecución de cada epoch superaba los 15 minutos, por lo que era prácticamente inviable hacer un entrenamiento apropiado que conllevase como mínimo 80 epoch (supondría 20 horas de procesamiento).

Por ello, el único experimento exitoso en este caso implicó el uso de 10 canciones del dataset. En este caso, se probaron los distintos tamaños de secuencia y de *batch*. Sin embargo, tanto el uso de un tamaño inferior a 100, como el uso de un tamaño de *batch* de 64, ofrecieron como resultado piezas musicales totalmente inconexas tanto armónicamente como rítmicamente. En cambio, con una configuración como la extraída del blog del autor (100 de tamaño secuencia y 128 de tamaño de *batch*) los resultados eran buenos. Dado que la fase de generación implica cierto proceso estocástico ya que se trabaja con probabilidades para seleccionar las notas, se realizaron varias generaciones. De entre ellas se seleccionó una como resultado final, la cual tomó 140 epoch de entrenamiento, alcanzando un error de 0'3 %. El resultado puede consultarse aquí: `https://soundcloud.com/juan-julian-cea-moran/lstm-10songs`

Otra de los experimentos consistió en variar ligeramente la arquitectura propuesta, sustituyendo las celdas LSTM por celdas GRU. El objetivo era comprobar cómo este cambio afectaba tanto a la calidad de los resultados, como a los requisitos computacionales durante el entrenamiento. Se pudo observar que en etapas tempranas del entrenamiento, el error disminuía más rápidamente que en el experimento con LSTM, estancándose en un valor más elevado que en el caso de las LST. Concretamente, en el caso de entrenamiento con 10 canciones del dataset, el error se estancó en 0'4 % en el epoch 60. Se presentan varios ejemplos generados usando 10 y 15 canciones como datos de entrenamiento. Ambas pueden encontrarse en los siguientes enlaces respectivamente: `https://soundcloud.com/juan-julian-cea-moran/gru-10songs` y `https://soundcloud.com/juan-julian-cea-moran/gru-15songs`

Este modelo demuestra cómo una arquitectura recurrente *sequence-vector*, en principio orientada a resolver problemas de clasificación, es igualmente capaz de generar piezas musicales creativas.

4.2 Modelo sequence-sequence (stateful)

El siguiente modelo que se probó fue el *sequence-sequence* visto en la sección 3.3. En este caso, el valor del tamaño de secuencia se ha mantenido igual a 100 para todas las pruebas realizadas.

Los primeros experimentos realizados han sido sobre la topología base. Se ha probado en este caso con 10 y 50 canciones del dataset. Cabe recordar que en este modelo el tamaño del *batch* siempre es igual a uno. Los mejores resultados se han obtenido con 10 canciones, apreciándose coherencia melódica y armónica a lo largo de toda la obra generada. En el caso del entrenamiento con 50 canciones, esta arquitectura demuestra ser igualmente capaz de generar armonía coherente con la melodía, aunque con peor resultado que en el caso anterior. Ambos resultados pueden encontrarse en la siguiente lista de reproducción: `https://soundcloud.com/juan-julian-cea-moran/sets/resultados-modelo-2`

En otro experimento, se ha probado sustituyendo las celdas GRU por LSTM. No se han podido obtener resultados significativos puesto que durante la fase de entrenamiento, el error no bajó de 6'7 % pese a probar con diferentes valores en los parámetros de la red.

Podemos decir que este modelo es un buen ejemplo de arquitectura de composición musical haciendo uso de las capacidades de las redes recurrentes y aprovechando las ventajas de las celdas GRU.

4.3 Modelo RNN bidireccional + attention

Inicialmente se realizaron pruebas sobre la topología base. En dichas pruebas, se comenzó entrenando la red con 10 canciones del dataset. De todos los modelos probados, este último es el que más tiempo ha llevado para reducir el error por debajo de 1 %, siendo el número de epoch superior a 170. Además de este entrenamiento tan lento, cabe destacar que los resultados generados carecían de sentido alguno y estaban basados en la repetición de una misma nota durante varios segundos para después cambiar de nota y continuar dicha repetición. Por ello, no se ha podido considerar esta topología como válida. Como primera alternativa, se consideró el cambio de las celdas LSTM por GRU, siendo el resultado similar y no consiguiendo una reducción significativa en el tiempo de entrenamiento.

Por esta razón, se ha probado modificando dicha arquitectura. La modificación realizada ha sido probada de nuevo con 10 canciones y el tiempo de entrenamiento se ha mantenido por encima de las 170 epoch. El resultado, pese a tener una estructura menos repetitiva apreciándose cambios de nota y acordes intercalados, seguía sin tener sentido armónico y melódico de ningún tipo. De nuevo, las pruebas cambiando LSTM por GRU arrojaron los mismos resultados.

Por esto se puede afirmar que este último modelo no ha sido capaz de generar ninguna muestra válida. Aún así, se adjuntan los ejemplos generados: `https://soundcloud.com/juan-julian-cea-moran/sets/resultados-modelo-3`.

5 Conclusiones y trabajo futuro

En esta tesis de fin de máster se ha tratado el tema de la composición musical automática por computador. Más concretamente, haciendo uso de uno de los tipos de modelos de *Deep Learning* más punteros en el campo: las redes neuronales recurrentes.

Se ha revisado las distintas propuestas que a lo largo de los años han sido significativas en este campo de estudio. Desde los primeros avances en composición algorítmica por computador, hasta las arquitecturas más modernas basadas en técnicas de Inteligencia Artificial. Además, se ha realizado un estudio sobre los datos con los que estos sistemas trabajan, parte fundamental para comprenderlos. Durante esta revisión, se ha hecho especial hincapié en las celdas tipo LSTM y GRU, dos variantes de las redes recurrentes tradicionales que permiten aprender secuencias de mayor tamaño y con un coste computacional más reducido. En concreto, se ha visto qué trabajos destacan en el uso de estas dos evoluciones de las RNN, entendiendo así cómo han contribuido a empujar esta tecnología hacia lo que hoy constituye el actual estado del arte en lo referente a composición musical automática. Además de estos dos grandes exponentes en lo que respecta a las redes neuronales recurrentes, se ha profundizado estudiando otras variantes, como las redes recurrentes bidireccionales o los mecanismos de *attention*.

5.1 Conclusiones experimentales

Una vez estudiados estos modelos y propuestas, se ha procedido a realizar algunas implementaciones con el objetivo de probar la eficacia de estos métodos para generar piezas creativas. En este sentido, se han implementado tres arquitecturas principalmente: modelo *sequence-vector* (*stateless*), modelo *sequence-sequence* (*stateful*) y modelo bidireccional + *attention*.

El primero de ellos destaca por su buena respuesta tanto con LSTM como con GRU,

siendo estas ultimas celdas mejores en cuanto a gasto de recursos computacionales se refiere. El segundo modelo también ofrece buenos resultados, pero al contrario que en el caso anterior, sólo entrena con celdas GRU. Además, este segundo modelo muestra estructuras rítmicas repetidas a lo largo de la obra, lo cual es muy positivo. Cabe destacar que mientras el primer modelo no era entrenable con más de 10 canciones del dataset (o 15 en el caso de usar GRU), el segundo ofrece resultados decentes entrenando con hasta 50 canciones. También hay que tener en cuenta, que en los resultados generados por el segundo modelo, en ocasiones se encuentran estructuras iguales a las obras usadas como entrenamiento, lo que denota cierto *overfitting* el cual podría ser solucionado incrementando el valor del *dropout*.

Sin embargo, existe un problema con los modelos propuestos en esta tesis. En concreto, con el formato seleccionado para representar los datos. Debido a que se ha decidido añadir la duración de las notas al final de su representación, el tamaño del vocabulario con el que los sistemas han trabajado se ha visto incrementado hasta alcanzar los 2200 o 2300 valores (dependiendo de las canciones seleccionadas). Esto supone un incremento en la cantidad de recursos computacionales necesarios, repercutiendo directamente en el número máximo de canciones que pueden usarse para entrenar las arquitecturas. También implica una peor representación del ritmo en las canciones generadas. Como puede verse en la gran mayoría de ejemplos, no existe un tempo concreto y estricto, sino que todo se basa en la duración de las diferentes notas.

5.2 Trabajo futuro

Con el fin de continuar con la investigación comenzada en esta tesis, se pueden proponer ciertas lineas de trabajo futuro:

- La principal línea de trabajo futuro que se extrae de las conclusiones vistas es la modificación de la representación de los datos. Sería deseable el uso de una representación que no conllevase tantas clases diferentes con las que las arquitecturas tuviesen que trabajar. Un buen ejemplo sería las representaciones tipo *piano roll*, muy usadas en la literatura. Este cambio, puede suponer que, intuitivamente, sea posible entrenar las diferentes arquitecturas con más canciones. Esto resulta en un mayor abanico de opciones de cara a entrenar los modelos con diferentes géneros musicales.

- Ligado al punto anterior, sería conveniente trabajar con métodos que codificasen directamente el tempo, subdividiendo los datos de entrada por compases. Esto resultaría en obras con una estructura musical más organizada, en contraposición con los ejemplos generados en esta tesis, donde puede verse falta de ritmo en general.

- En esta tesis solo se ha probado a entrenar las redes con canciones de música clásica, así que otra tarea pendiente de cara al futuro es probar la capacidad de estos modelos de generar música de otros estilos. Sería además conveniente combinar datasets de diferentes géneros musicales y ver si los resultados generados son satisfactorios. En este sentido, una idea muy interesante sería mezclar datasets de géneros con relación en la historia de la música y ver si el resultado obtenido se corresponde con los subgéneros derivados.

- Por ultimo, la línea de trabajo más evidente sería probar distintas arquitecturas basándose en lo aprendido en esta tesis. Por ejemplo, sería interesante el diseño de un sistema capaz de aprender diferentes pistas con diferentes instrumentos cada una y combinarlos para generar obras más completas.

Bibliografía

[1] Christopher Ariza. The interrogator as critic: The turing test and the evaluation of generative music systems. *Computer Music Journal*, 33(2):48–70, 2009.

[2] Edward P Asmus. Music assessment concepts: A discussion of assessment concepts and models for student assessment introduces this special focus issue. *Music educators journal*, 86(2):19–24, 1999.

[3] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[4] Jean-Pierre Briot, Gaëtan Hadjeres, and François-David Pachet. Deep learning techniques for music generation–a survey. *arXiv preprint arXiv:1709.01620*, 2017.

[5] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[6] Ching-Hua Chuan and Dorien Herremans. Modeling temporal tonal relations in polyphonic music through deep networks with a novel image-based representation. In *Thirty-second AAAI conference on artificial intelligence*, 2018.

[7] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[8] Michael Scott Cuthbert. music21 - https://github.com/cuthbertlab/music21.

[9] Hao-Wen Dong, Wen-Yi Hsiao, Li-Chia Yang, and Yi-Hsuan Yang. Musegan: Multi-track sequential generative adversarial networks for symbolic music generation

and accompaniment. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[10] Douglas Eck and Juergen Schmidhuber. A first look at music composition using lstm recurrent neural networks. *Istituto Dalle Molle Di Studi Sull Intelligenza Artificiale*, 103:48, 2002.

[11] freemidi.org. Free midi - https://www.freemidi.org/.

[12] Aurélien Géron. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media, 2019.

[13] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[14] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[15] Google. Magenta-tensorflow datasets - https://magenta.tensorflow.org/datasets/.

[16] Klaus Greff, Rupesh K Srivastava, Jan Koutník, Bas R Steunebrink, and Jürgen Schmidhuber. Lstm: A search space odyssey. *IEEE transactions on neural networks and learning systems*, 28(10):2222–2232, 2016.

[17] Lejaren Arthur Hiller and Leonard M Isaacson. *Experimental Music; Composition with an electronic computer*. Greenwood Publishing Group Inc., 1979.

[18] Sepp Hochreiter. Untersuchungen zu dynamischen neuronalen netzen. *Diploma, Technische Universität München*, 91(1), 1991.

[19] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[20] Thom Holmes. *Electronic and experimental music: technology, music, and culture*. Routledge, 2012.

[21] John J Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences*, 79(8):2554–2558, 1982.

[22] Cheng-Zhi Anna Huang, Ashish Vaswani, Jakob Uszkoreit, Ian Simon, Curtis Hawthorne, Noam Shazeer, Andrew M Dai, Matthew D Hoffman, Monica Dinculescu, and Douglas Eck. Music transformer: Generating music with long-term structure. In *International Conference on Learning Representations*, 2018.

[23] Alex Issa. Generating original classical music with an lstm neural network and attention - https://medium.com/@alexissa122/generating-original-classical-music-with-an-lstm-neural-network-and-attention-abf03f9ddcb4, 2019.

[24] Daniel Johnson. Composing music with recurrent neural networks - http://www.hexahedria.com/2015/08/03/composing-music-with-recurrent-neural-networks/.

[25] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4401–4410, 2019.

[26] Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Analyzing and improving the image quality of stylegan. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8110–8119, 2020.

[27] Athanasius Kircher and Ulf Scharlau. *Musurgia universalis*, volume 2. Olms, 2006.

[28] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[29] Bernd Krueger. Classical piano midi page - http://www.piano-midi.de/.

[30] Yann LeCun, Yoshua Bengio, et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.

[31] Yann LeCun, Bernhard E Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne E Hubbard, and Lawrence D Jackel. Handwritten digit recognition with a back-propagation network. In *Advances in neural information processing systems*, pages 396–404, 1990.

[32] Xia Liang, Junmin Wu, and Jing Cao. Midi-sandwich2: Rnn-based hierarchical multi-modal fusion generation vae networks for multi-track symbolic music generation. *arXiv preprint arXiv:1909.03522*, 2019.

[33] Hyungui Lim, Seungyeon Rhyu, and Kyogu Lee. Chord generation from symbolic melody using blstm networks. *arXiv preprint arXiv:1712.01011*, 2017.

[34] Sephora Madjiheurem, Lizhen Qu, and Christian Walder. Chord2vec: Learning musical chord embeddings. In *Proceedings of the constructive machine learning workshop at 30th conference on neural information processing systems (NIPS2016), Barcelona, Spain*, 2016.

[35] Max V Mathews and F Richard Moore. Groove—a program to compose, store, and edit functions of time. *Communications of the ACM*, 13(12):715–721, 1970.

[36] MIDIWORLD. midiworld - http://www.midiworld.com/.

[37] MIDI Manufacturers Association (MMA). Midi association - https://www.midi.org/.

[38] Michael C Mozer. Neural network music composition by prediction: Exploring the benefits of psychoacoustic constraints and multi-scale processing. *Connection Science*, 6(2-3):247–280, 1994.

[39] Aran Nayebi and Matt Vitelli. Gruv: Algorithmic music generation using recurrent neural networks. *Course CS224D: Deep Learning for Natural Language Processing (Stanford)*, 2015.

[40] Gerhard Nierhaus. *Algorithmic composition: paradigms of automated music generation*. Springer Science & Business Media, 2009.

[41] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.

[42] Mirco Ravanelli, Philemon Brakel, Maurizio Omologo, and Yoshua Bengio. Light gated recurrent units for speech recognition. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 2(2):92–102, 2018.

[43] Curtis Roads and Max Mathews. Interview with max mathews. *Computer Music Journal*, 4(4):15–22, 1980.

[44] Adam Roberts, Jesse Engel, Colin Raffel, Curtis Hawthorne, and Douglas Eck. A hierarchical latent vector model for learning long-term structure in music. *arXiv preprint arXiv:1803.05428*, 2018.

[45] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[46] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

[47] Mike Schuster and Kuldip K Paliwal. Bidirectional recurrent neural networks. *IEEE transactions on Signal Processing*, 45(11):2673–2681, 1997.

[48] Four Score and More. Musorepo: a directory of resources for computational musicology - https://fourscoreandmore.org/musorepo/.

[49] Sigurður Skúli. How to generate music using a lstm neural network in keras - https://towardsdatascience.com/how-to-generate-music-using-a-lstm-neural-network-in-keras-68786834d4c5.

[50] Bob L Sturm, Joao Felipe Santos, Oded Ben-Tal, and Iryna Korshunova. Music transcription modelling and composition using deep learning. *arXiv preprint arXiv:1604.08723*, 2016.

[51] Peter M Todd. A connectionist approach to algorithmic composition. *Computer Music Journal*, 13(4):27–43, 1989.

[52] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[53] Chris Walshaw. abc notation - http://www.abcnotation.org/.

[54] Ronald J Williams and David Zipser. Gradient-based learning algorithms for recurrent. *Backpropagation: Theory, architectures, and applications*, 433, 1995.

[55] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *International conference on machine learning*, pages 2048–2057, 2015.

[56] Li-Chia Yang and Alexander Lerch. On the evaluation of generative models in music. *Neural Computing and Applications*, pages 1–12, 2018.

[57] Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy. Hierarchical attention networks for document classification. In *Proceedings of the 2016 conference of the North American chapter of the association for computational linguistics: human language technologies*, pages 1480–1489, 2016.

[58] Yichao Zhou, Wei Chu, Sam Young, and Xin Chen. Bandnet: A neural network-based, multi-instrument beatles-style midi music composition machine. *arXiv preprint arXiv:1812.07126*, 2018.