E.T.S. de Ingeniería Industrial, Informática
y de Telecomunicación

Aprendizaje profundo aplicado a la generación y transcripción de música

Grado en Ingeniería Informática

Trabajo Fin de Grado

Ibon Escartín Marcotegui

Daniel Paternain Dallo

Pamplona, 27 de octubre de 2020

upna
Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa

Resumen

La música es un lenguaje que permite la comunicación de sensaciones y sentimientos de forma no verbal. Como lenguaje, está formada por un conjunto de reglas que permiten construir música que evoque una sensación u otra y si no se aplican correctamente, el resultado puede ser un conjunto de sonidos sin sentido. Este trabajo trata de buscar y comparar las distintas formas en las que se puede representar la música y cuales de ellas son las mas adecuadas para la generación de música. Además se mencionan algunas otras representaciones que a pesar de no dar muy buenos resultados para esta tarea, son útiles para otras tareas como la transcripción de música.

Palabras clave

aprendizaje profundo, redes neuronales recurrentes, NLP, música

Índice

1 Introducción y objetivos....1

1.1 Objetivos....2

2 Estándares de codificación de música....2

2.1 Notación ABC....3

2.1.1 Generación carácter a carácter....11

2.1.2 Generación token a token....11

2.2 MIDI....12

2.2.1 Piano roll....15

2.2.2 Agrupamiento de notas....17

2.2.3 Codificación de Performance RNN....18

2.3 Señal de audio....20

3 Redes neuronales recurrentes....22

3.1 LSTM....25

3.2 GRU....27

3.3 Generación de secuencias mediante redes neuronales recurrentes....28

4 Redes neuronales profundas para la generación de música....29

4.1 Generación carácter a carácter empleando notación ABC....30

4.2 Generación token a token con notación ABC....34

4.3 Generación MIDI simple....37

4.4 Generación empleando Performance RNN....39

4.5 Generación empleando audio....42

5 Otras representaciones....43

5.1 Transformada de Fourier....43

5.2 Espectrograma....44

6 Conclusiones y líneas futuras....44

7 Bibliografía....47

1 Introducción y objetivos

Hace años, las tareas creativas escapaban del alcance de cualquier tipo de inteligencia artificial. Las tareas se resolvían prácticamente por fuerza bruta, estudiando todas las posibilidades y seleccionando la más prometedora. Las aplicaciones de este tipo de algoritmos quedaban limitadas a tareas como la resolución de laberintos o el juego del ajedrez. Este tipo de algoritmos eran fuertemente dependientes de su programación, como es el caso de Deep Blue, el ordenador que en 1997 derrotó al entonces campeón mundial de ajedrez Garri Kaspárov, que estudiaba millones de posibles movimientos por segundo y, mediante una función de evaluación que tenía en cuenta el valor y posición de cada pieza, seleccionaba el movimiento adecuado en cada momento.

Una forma de eliminar esta dependencia y evitar que las reglas para resolver el problema estuvieran codificadas en el propio algoritmo es el empleo de redes neuronales que, a pesar de que existen desde mucho antes, no ha sido posible sacarles todo el partido hasta hace relativamente poco. Con el avance de la tecnología, la capacidad de procesamiento ha aumentado de forma exponencial, facilitando el entrenamiento de redes más complejas, las redes neuronales profundas, capaces de resolver problemas que anteriormente tenían un coste computacional elevado o eran directamente intratables.

Como resultado de este avance, se han ido presentando nuevas arquitecturas capaces de realizar tareas que antes podían parecer imposibles, como las que permiten la descripción de imágenes empleando lenguaje natural e incluso el proceso inverso, la obtención de una imagen a partir de su descripción, o CycleGAN, que permite transformar una imagen en otra en la que por ejemplo tomando la foto de un paisaje en verano se obtiene el mismo paisaje en invierno, cubierto de nieve, o la transferencia de estilo de un pintor, haciendo que una foto pueda parecer un cuadro pintado por Van Gogh o StyleGAN que es capaz de generar imágenes altamente realistas, entre otros muchos ejemplos.

En casos como estos últimos se puede ver como el espectro de aplicaciones se va ensanchando y no solo se dedica a resolver problemas aplicando una serie de reglas que pueden incluso haber sido predefinidas, sino que es capaz de hallar esos patrones y reglas partiendo de un conjunto de datos inicial y generar datos nuevos con los mismos patrones y reglas, entrando en un campo que tradicionalmente se ha asociado a los humanos, como es el de la creatividad.

Los datos que se le presentan a estas redes pueden ser representados de maneras muy distintas, pudiendo obtener un rendimiento muy bajo en arquitecturas muy complejas

si la representación es mala y un muy buen rendimiento en arquitecturas mucho más simples si se elige una representación adecuada.

1.1 Objetivos

El objetivo de este trabajo fin de grado es realizar un estudio y análisis en profundidad de diferentes formas de representación de la música con el objetivo de aplicar técnicas de aprendizaje profundo. En concreto, nos centraremos en la generación o composición automática de música.

Para alcanzar este objetivo se completarán los siguientes subobjetivos:

- Analizar las redes neuronales recurrentes, un tipo de redes que son capaces de tratar datos que poseen un carácter temporal.
- Estudiar las diferentes formas que existen para codificar la música, como pueden ser la notación ABC o MIDI, analizar cómo funcionan y comparar sus ventajas y desventajas.
- Buscar y comparar distintas arquitecturas de red para la generación de música.
- Explorar las posibles aplicaciones de las representaciones menos aptas para la generación de música.

2 Estándares de codificación de música

Las posibles representaciones de la música se pueden agrupar en dos categorías.

- Representaciones semánticas, que emplean un lenguaje descriptivo de la música. Estas codificaciones abstraen la descripción física del sonido y suelen incluir símbolos que indican la duración de la nota, el tono, el tempo, la repetición de secciones o el nombre del instrumento que la interpreta. Además, estas representaciones emplean una serie de reglas para combinar estos símbolos.
- Representaciones no semánticas, en las que la descripción de la música se realiza a nivel físico. En estas representaciones se codifican magnitudes físicas como la amplitud de la onda o la frecuencia del sonido a lo largo del tiempo.

Existen multitud de codificaciones posibles de ambas categorías. Entre las representaciones simbólicas más populares se encuentran MusicXML, un formato que permite codificar las partituras en XML para poder compartirlas entre distintos editores de partituras incompatibles entre sí al emplear cada uno un formato propio y MIDI, un protocolo diseñado para facilitar la comunicación entre ordenadores e instrumentos electrónicos y virtuales. La representación no semántica más habitual es el formato wav, que codifica la señal de audio sin ningún tipo de compresión. El resto

de representaciones de esta categoría suelen ser transformaciones sobre la señal de audio como las distintas transformadas de Fourier o el espectrograma.

Para generar música, a priori parece más sencillo tratar con las representaciones simbólicas, porque se pueden encontrar patrones fácilmente en cualquier conjunto de símbolos combinados empleando un conjunto de reglas.

De todas las representaciones posibles, en este trabajo se han evaluado:

- la notación ABC, que a pesar de no ser muy popular permite una representación sencilla de muchos conceptos musicales. A diferencia de MusicXML no es estricto a la hora de ser interpretado por una máquina, pues los símbolos que se añadan en lugares que no corresponde son ignorados por la mayoría de intérpretes de este tipo de notación mientras que los intérpretes de MusicXML únicamente son capaces de interpretar XML válido.
- La codificación MIDI, porque es probablemente la representación más popular a la hora de representar música en un formato interpretable por máquinas y humanos y la información está codificada con una mayor abstracción que con la notación ABC, permitiendo obtener resultados igual de complejos con un número menor de símbolos.

El resto de representaciones no son del todo adecuadas para esta tarea, pues como veremos más adelante y debido al alto número de muestras que hay que tomar para que una señal de audio no tenga pérdidas, la longitud de las secuencias es enorme y es complicado abstraer conceptos como ritmo, melodía o timbre. Es por ello que este tipo de representaciones son generalmente desechadas para esta tarea y son empleadas para otras tareas, como pueden ser la clasificación y la transcripción. Aun así, existen arquitectas como WaveNet [1] que es capaz de generar audio muestra a muestra solventando el problema de las dependencias a muy largo plazo tomando convoluciones temporales o SampleRNN [2], que obtiene el valor de cada muestra como el producto de la probabilidad condicional de la muestra en base a todas las muestras anteriores. Es por ello que también se evaluará la representación en forma de audio crudo para tratar de generar música empleando WaveNet.

2.1 Notación ABC

La notación ABC es un tipo de notación basada en la notación anglosajona, empleando letras desde la A hasta la G para nombrar las notas y añadiendo una serie de símbolos adicionales en forma de caracteres ASCII para representar la duración de cada nota, sus alteraciones, barras de compás etc., además de contener cabeceras que permiten añadir el compás, el tempo, el título de la obra, o el nombre del autor por ejemplo. Fue desarrollada con la finalidad de poder tener la capacidad de editar música con

cualquier programa capaz de manipular texto plano, además de tratar de hacer que sea fácilmente legible por humanos.

Un fichero ABC está formado por una sección de cabeceras (*file header*) y una sección en la que aparecen cada una de las piezas (*tune body*), pues puede haber múltiples piezas en un mismo fichero, cada una de ellas con sus propias cabeceras (*tune header*). Las cabeceras de cada pieza contienen metadatos como el título, el autor, tempo, compás o incluso comentarios.

La sección de cabeceras está formada por una lista de campos de la forma *símbolo: valor*, donde *símbolo* es uno de los posibles símbolos mostrados en la tabla 1 y *valor* es el valor asociado al concepto representado por ese símbolo.

Si en un fichero existen varias piezas, estas toman todas por defecto los valores de cabeceras definidos en la sección de cabeceras del fichero. Si una pieza tiene cabeceras que entran en conflicto con las definidas en la sección cabeceras global, tienen prioridad las propias antes que las globales.

Los campos que forman las cabeceras pueden aparecer además de en las cabeceras del fichero o de la pieza en el propio *tune body* o a lo largo de la pieza entre los símbolos musicales. Esta última forma de aparición es denominada *inline*.

Símbolo	**Puede aparecer en**	**Descripción**
A	file header tune header	Información sobre el origen de la pieza. Se recomienda usar el campo O en su lugar
B	file header tune header	Nombre del libro impreso de origen de la pieza
C	file header tune header	Nombre del compositor de la pieza
D	file header tune header	Nombre de un disco en el que se puede escuchar la pieza
F	file header tune header	URL del sitio web desde el que se puede descargar el fichero ABC

Símbolo	Puede aparecer en	Descripción
G	file header tune header	Valor por el que se agrupan los ficheros ABC en las bases de datos
H	file header tune header	Historia de la pieza
I	file header tune header tune body inline	Modifica como debe ser interpretado el código ABC
K	tune header (al final) tune body inline	Tonalidad de la pieza
L	file header tune header tune body inline	Duración de la nota por defecto
M	file header tune header tune body inline	Compás
N	file header tune header tune body inline	Notas sobre la pieza
O	file header tune header	Origen de la pieza

Símbolo	Puede aparecer en	Descripción
P	tune header tune body inline	Marca las partes que definen el orden en el que se toca la pieza
Q	tune header tune body inline	Tempo de la pieza
R	file header tune header tune body inline	Ritmo de la pieza. Ofrece algo más de información sobre la pieza al músico, además de poder ser usado para etiquetar la pieza en bases de datos
r	file header tune header tune body inline	Comentario
S	file header tune header	Fuente de la que se ha tomado la pieza
s	tune body	Línea de símbolos. Permite añadir símbolos que clarifican o añaden información en caso de que los que hubiera no fueran fáciles de leer o requieran algún símbolo sobre la dinámica de la pieza
T	tune header (en segunda posición) tune body	Título de la pieza

Símbolo	Puede aparecer en	Descripción
U	file header tune header tune body inline	Símbolo definido por el usuario. Permite realizar asociaciones entre caracteres y símbolos preexistentes generalmente con la finalidad de acortar la longitud de sus nombres y facilitar la lectura y la escritura
V	tune header tune body inline	Permite definir múltiples voces
W	tune header tune body inline	Letra de la pieza. Sin importar donde se use este campo, la letra siempre aparecerá al final
w	tune body	Letra de la pieza alineada con las notas
X	tune header (en primera posición)	Número de secuencia. Identifica de forma única a la pieza en caso de que haya más de una pieza en el mismo fichero
Z	file header tune header	Datos del autor de la transcripción

Tabla 1: Campos de un fichero ABC[1]

A continuación de la cabecera viene la sección que contiene la música propiamente dicha. Esta sección está dividida en compases, una división que se puede realizar empleando símbolos distintos con una funcionalidad distinta cada uno de ellos.

1 Tablas 1 y 2 modificadas a partir de las originales de http://abcnotation.com/wiki/abc:standard:v2.1

<table>
<tr><th>Símbolo</th><th>Significado</th></tr>
<tr><td>|</td><td>Barra de compás</td></tr>
<tr><td>|]</td><td>Barra doble final</td></tr>
<tr><td>||</td><td>Barra doble</td></tr>
<tr><td>[|</td><td>Barra doble inicial</td></tr>
<tr><td>|:</td><td>Comienzo de sección repetida</td></tr>
<tr><td>:|</td><td>Fin de sección repetida</td></tr>
<tr><td>::</td><td>Comienzo y fin de dos secciones repetidas</td></tr>
</table>

Tabla 2: Tipos de barras de compás

Dentro de cada compás se añaden las notas y los silencios. Cada uno de los tonos que representa a cada nota se codifica mediante la letra correspondiente en el sistema anglosajón, comenzando con mayúsculas para las notas más graves y pasando a minúsculas para las más agudas. Se puede añadir además una o más comas a continuación para bajarlas una octava por cada coma que se añade o un apóstrofe o más para subir la tonalidad una octava también por cada apóstrofe añadido.

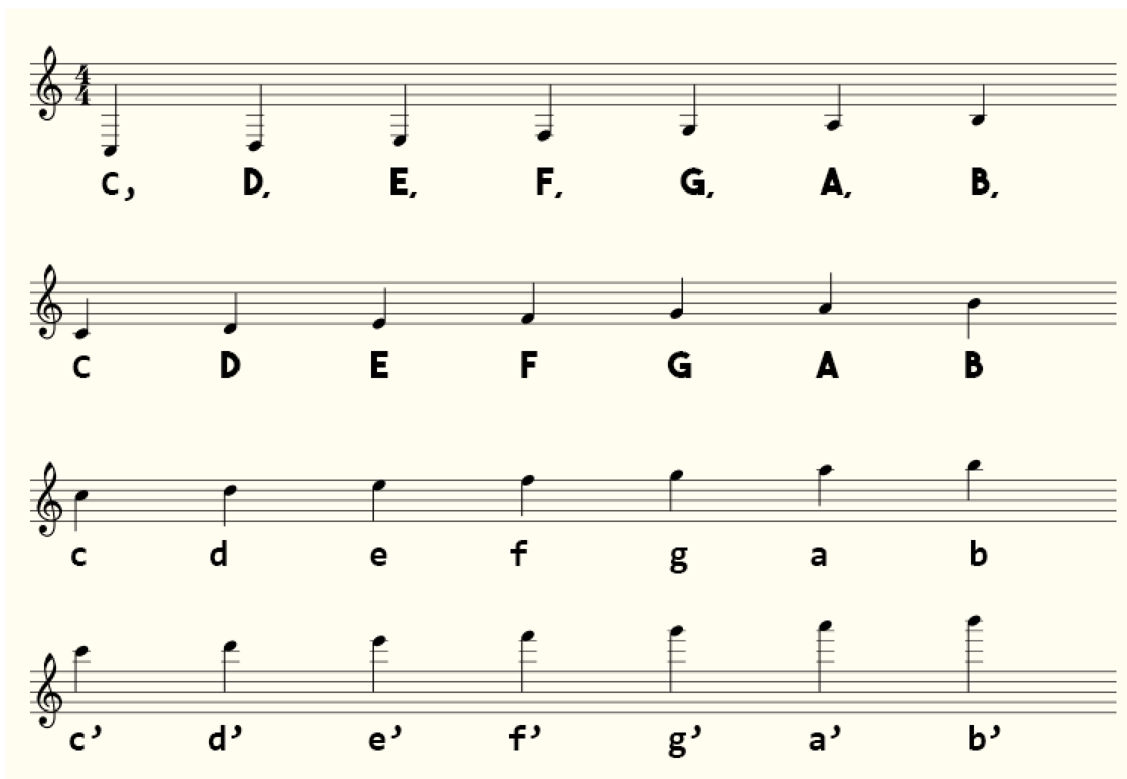

Figura 1: Codificación de los tonos. Se pueden añadir comas y apostrofes adicionales para representar octavas más bajas/altas

Para añadir alteraciones a las notas se emplean los símbolos ^, = y _ que se añaden antes de la nota y representan el sostenido, el becuadro y el bemol. Para añadir dobles bemoles y sostenidos se repite el símbolo dos veces __ y ^^

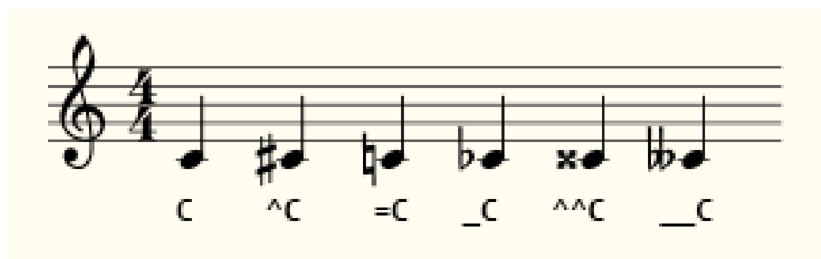

Figura 2: Alteraciones en notación ABC

Si se desea modificar la duración de una nota, es necesario añadirle a continuación del símbolo que marca su tono un número que indica por cuanto se multiplica la duración por defecto, fijada en las cabeceras. En la figura 3 se puede ver como la misma cadena puede dar resultados diferentes si se cambia la duración por defecto.

Este número puede ser un número entero o una fracción y en el caso de las fracciones, si el numerador es 1 puede ser omitido. Así por ejemplo, si en una pieza la duración por defecto es 1/8 o lo que es lo mismo, una corchea, la nota `C` tendrá la duración de una corchea mientras que la nota `C2` tendrá la duración de una negra, `C/2` o `C1/2`

una semicorchea y C5/2 una negra con puntillo. El silencio se representa con la letra z y su duración se modifica de la misma forma.

Figura 3: Duraciones en notación ABC. *En ambos casos la cadena de caracteres en notación ABC es la misma (C/8 C/4 C/2 C C2 C3 C4 C8), pero la duración de la cabecera es distinta.*

Si la pieza en cuestión es únicamente una melodía, esta suele incluir acordes de acompañamiento. Estos acordes se encuentran siempre entre comillas dobles y están formados por una letra mayúscula correspondiente a una de las notas en cifrado anglosajón, un carácter que indica una alteración (opcional) que puede tomar el valor b para bemol y # para sostenido, el tipo de acorde (también opcional) que puede ser uno de los valores de la tabla 3 y finalmente y de forma también opcional otro carácter en mayúsculas precedido por una barra / que representa la nota en el bajo. Se pueden formar acordes que no sean de acompañamiento agrupando las notas que forman el acorde entre corchetes [].

Tipo de acorde en notación ABC	**Tipo de acorde**
m o min	Acorde menor
maj	Acorde mayor
dim	Acorde disminuido
aug o +	Acorde aumentado
sus	Acorde suspendido
7, 9, ...	Acorde con séptima, novena, ...

Tabla 3: Tipos de acordes

```
X: 8
T:Atholl Highlanders
% Nottingham Music Database
P:AABBCCDD
S:Kevin Briggs, via EF
M:6/8
K:A
P:A
|:c/2d/2|"A"e3 ecA|"A"ecA "E7"Bcd|"A"e3 ecA|"D"Bcd "E7"cBA|
"A"e3 ecA|"A"ecA "E7"Bcd|"A"e/2a3/2e "D"fed|"E7"cdB "A"A2:|
P:B
|:e|"A"Ace Ace|"D"Adf Adf|"A"Ace Ace|"Bm"Bcd "E7"c2B|
"A"Ace Ace|"D"Adf Adf|"A"e/2a3/2e "D"fed|"E7"cdB "A"A2:|
P:C
|:e|"A"aee edc|"A"aee edc|"A"aee edc|"D"Bcd "E7"cBA|
"A"aee edc|"A"aee edc|"A"aee fed|"E7"cdB "A"A2:|
P:D
|:A/2B/2|"A"cAc cAc|"D"dBd dBd|"A"cAc cAc|"G"B=GB BGB|
"A"cAc cAc|"E7"dBd dBd|"A"e/2a3/2e "D"fed|"E7"cdB "A"A2:|
```

Texto 1: Atholl Highlanders, una canción folk escocesa en notación ABC

2.1.1 Generación carácter a carácter

La representación más simple consiste en tratar el texto como una secuencia de caracteres e intentar predecir el siguiente carácter dada una secuencia concreta. Este es el método más simple, y es el empleado en muchos casos para la generación de texto.

Para ello, se divide el texto en caracteres, a los que se les asigna un número. Esta relación carácter-número se almacena en un diccionario en el que cada par clave-valor es un carácter y un número. Es necesario crear un diccionario con la relación inversa, guardando las asociaciones número-carácter para poder deshacer el proceso una vez se genere el texto.

El siguiente paso consiste en vectorizar las secuencias, transformando cada secuencia de caracteres en una lista de números en la que cada carácter es convertido en un número empleando el primero de los diccionarios mencionados. Una vez se dispone de las listas de números, se convierten en matrices empleando una codificación one-hot.

2.1.2 Generación token a token

La generación token a token es similar a la generación carácter a carácter, a diferencia de que la unidad mínima de información en este caso son los tokens ABC y no los caracteres. Para ello es necesario emplear un parser de notación ABC para extraer todos los tokens de un texto y repetir el proceso anterior de asignarle un número a cada token, transformar las secuencias de tokens en listas de números y convertirlas en último lugar en matrices one-hot.

La única diferencia con la generación carácter a carácter es la necesidad de eliminar algunos tokens únicos en cada pieza y que no aportan ningún tipo de información para esta tarea como son las cabeceras de número de secuencia, título de la obra y autor, además cualquier otra cabecera de metadatos.

2.2 MIDI

MIDI son las siglas para *Musical Instrument Digital Interface,* un estándar que permite la comunicación entre instrumentos musicales y ordenadores. Este estándar permite que los dispositivos se comuniquen entre sí mediante el envío de mensajes. La especificación del estándar define 128 tipos distintos de mensajes que permiten enviar y recibir información sobre cuánto se ha pisado un pedal o si se ha pulsado o no una tecla. Además de para comunicación en tiempo real, estos mismos mensajes se pueden codificar en un fichero para ser posteriormente reproducidos de la misma forma que si hubieran sido emitidos al instante.

Los ficheros MIDI están codificados como archivos binarios, compuestos por una cabecera en la que se describe la estructura general del archivo y una serie de pistas, cada una de ellas conteniendo los mensajes individuales (tempo, inicio de una nota, fin de una nota…)

Estos mensajes se pueden enviar por cualquiera de los 16 canales disponibles, permitiendo mediante ellos que una única conexión controle diferentes instrumentos al asignar cada uno a un canal. Así se puede poner por ejemplo, una guitarra en el canal 1, una trompeta en el canal 2 y una batería en el canal 3 y desde el controlador MIDI tocar cada instrumento de forma individual únicamente modificando el canal al que se envían los mensajes. Cada uno de los mensajes se envía en un instante de tiempo, marcado por el número de *ticks* MIDI. Un *tick* MIDI es una unidad de tiempo definida en la cabecera de cada fichero, pues su duración es variable en función del tempo. El tiempo en el que sale cada mensaje es relativo al último mensaje. De esta forma, se pueden enviar varios mensajes que aunque en la pista aparezcan de forma secuencial, si el tiempo relativo entre ellos es 0 se interpreta que todos los eventos han ocurrido a la vez, permitiendo por ejemplo que suenen múltiples notas simultáneamente.

A diferencia de como se define típicamente, el tempo en los ficheros MIDI se representa en microsegundos por pulso. La duración de un *tick* viene dada por la fórmula de la figura 4.

$$1\,tick = \frac{microsegundos\ por\ pulso}{60}$$

Figura 4: Fórmula para calcular la duración de 1 tick en microsegundos

Esta información está incluida al final de la cabecera del archivo. La cabecera comienza con los bytes `4D 54 68 64` (correspondientes a los caracteres `MThd` en ASCII) que identifican al archivo como un fichero MIDI. A continuación se especifica el tamaño de la cabecera codificado como un entero de 4 bytes. Los siguientes 2 bytes se emplean para codificar el tipo de archivo MIDI. Existen 3 tipos de archivo: Los archivos con una única pista, los archivos multipista síncronos y los archivos multipista asíncronos, codificados como 0, 1 y 2 respectivamente. Los archivos con una pista, como su propio nombre indica, solo contienen una pista, mientras que los archivos multipista síncronos y asíncronos se diferencian en cuando comienzan a reproducirse sus pistas: Se reproducen todas las pistas simultáneamente en los ficheros síncronos, no siendo necesariamente así en los asíncronos.

Los siguientes 4 bytes contienen el número de pistas y la duración de una negra en ticks. Se asignan dos bytes a cada una de las magnitudes.

A continuación vienen las pistas. Cada una de las pistas consiste en una cabecera similar a la que se ha visto antes, con cuatro bytes fijos (`4D 54 72 6B` en este caso, `MTrk` en ASCII) y 4 bytes más en los que se codifica la longitud de la pista en bytes. Estas pistas pueden contener tantos mensajes MIDI como se desee.

A cada uno de los mensajes le precede el delta-time, el tiempo transcurrido desde el último evento en ticks. Estos mensajes son idénticos a los mensajes que envían los dispositivos compatibles con el estándar MIDI. La duración se codifica como un entero de tamaño variable. De cada uno de los bytes empleados para codificarlo, el bit más significativo toma siempre el valor 1, mientras que los 7 restantes son empleados para codificar el número. En caso de ser el último byte, el bit más significativo toma el valor de 0. Esto permite emplear tantos bytes como sean necesarios para representar el número si es un número grande y a la vez emplear un número reducido de bytes si el número es pequeño.

El mensaje está compuesto por un byte llamado byte de comando y una serie de bytes de datos diferentes para cada comando. El bit más significativo del comando toma siempre el valor 1, destinando los 3 siguientes bits para codificar el comando y los 4 bits restantes para codificar el canal a través del que se enviará. Los bytes de datos en cambio, tienen el bit más significativo con valor 0. El estándar MIDI define 8 comandos distintos.

Valor (binario)	Comando	Descripción
000	Note off	Indica cuando una nota ha finalizado. Le siguen dos bytes de datos, uno con el número de nota y otro con la velocidad[2].
001	Note on	Indica cuando una nota ha comenzado. Los bytes de datos tienen el mismo formato que en el comando note off
010	Key after-touch	Permite modificar la presión con la que se ha pulsado una tecla mientras la tecla se mantiene pulsada
011	Control change	Mensaje de control. Hay 128 tipos de mensajes de control distintos, y permiten controlar desde los pedales de un piano hasta la intensidad con la que se sopla un instrumento de viento
100	Program change	Modifica el instrumento que está sonando. El estándar MIDI define 128 instrumentos distintos

2 El término velocidad puede ser confundido con el tempo o la “velocidad” a la que se interpreta la pieza si no se explica en el contexto en el que se emplea en el estándar MIDI. Este estándar utiliza la palabra velocidad para dar nombre a la fuerza o intensidad con la que suena una nota.

Valor (binario)	Comando	Descripción
101	Channel after-touch	Permite modificar la presión con la que se ha pulsado un grupo de teclas, manteniendo la misma presión para todo el grupo sin importar las presiones individuales de cada tecla
110	Pitch-wheel change	Modifica el tono de la nota que está sonando en función de la posición de la rueda de tono
111	System	Permite a los fabricantes de dispositivos compatibles con el estándar definir sus propios mensajes

Tabla 4: Comandos definidos en el estándar MIDI

Como se puede comprobar, este formato no es muy amigable para ser empleado directamente como representación de los datos. Su ventaja es que es sencillamente transformable a otros formatos más cómodos, como por ejemplo *piano roll*.

2.2.1 Piano roll

Esta es la forma más común de representar los archivos MIDI. Se emplea habitualmente en software de edición de música o *DAW* (siglas para *Digital Audio Workstation*), pues permite visualizar de una forma cómoda el tono y la duración de cada una de las notas. Las notas se representan por su altura y duración, mostrándose más arriba en el eje vertical cuanto más alto sea su tono, y siendo más largas o cortas en el eje horizontal en función de su duración. Al ser representados de esta manera se visualiza fácilmente en qué momento ocurren los eventos *note on* y *note off* en cada una de las notas (al principio y al final de cada uno de los rectángulos, respectivamente), y habitualmente se puede ver también la velocidad de cada una de ellas.

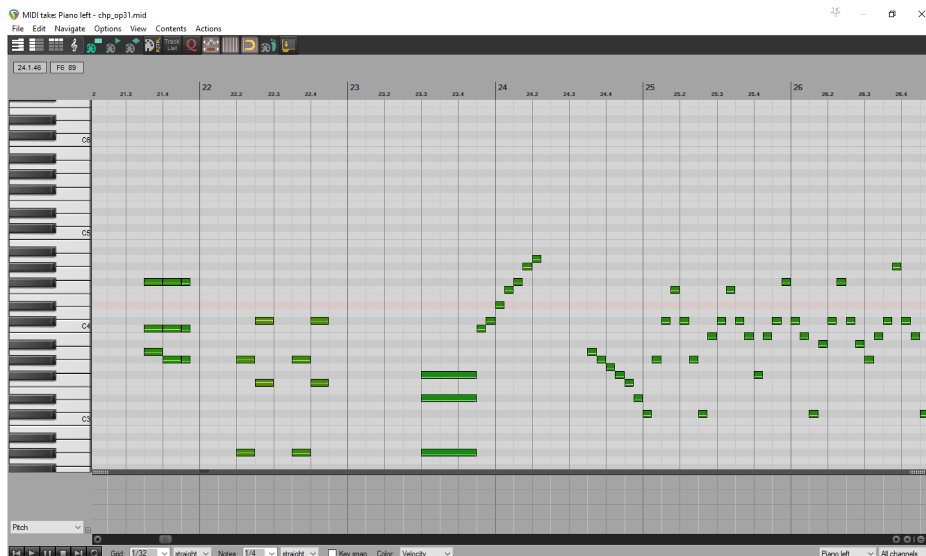

Figura 5: Archivo MIDI tal y como se ve en Reaper[3]

Este formato nace en las pianolas, que poseen un rollo de papel perforado que marca cuándo y por cuánto tiempo se debe tocar cada una de las notas.

Figura 6: Fragmento de un rollo de pianola[4]

Para obtener la representación en forma de *piano roll* de un fichero MIDI se pueden aplicar distintos algoritmos en función de la información que se desee conservar. Su principal carencia es la enorme dispersión de los datos, puesto que la gran mayoría de la matriz contiene valores nulos al corresponder cada una de las columnas de la matriz con un tick MIDI. Algunas de las formas de reducir el tamaño de esta matriz incluyen el muestreo de la matriz, seleccionando columnas a intervalos regulares de tiempo, con la consecuente pérdida de información que esto conlleva, pues una frecuencia de

3 Sitio web: https://www.reaper.fm/

4 Rollo de “Chants d'Espagne, op. 232. Nº 4, Córdoba” de Isaac Albéniz. Imagen original por Biblioteca Nacional de España bajo licencia CC BY-NC-SA 4.0

muestreo muy alta puede eliminar por completo las notas de menor duración. Otra de las posibilidades de reducir la dispersión es reducir el rango de notas posible a la tesitura del instrumento o transformarla a otro tipo de codificación relacionada con esta como el agrupamiento de notas o la codificación empleada por arquitecturas como Performance RNN [3].

2.2.2 Agrupamiento de notas

Para transformar un fichero MIDI a esta representación, se avanza entre eventos *note on*, ignorando la velocidad de estos y el resto de mensajes. De esta forma se simplifica mucho el problema a cambio de perder características como el ritmo y las intensidades con las que suena cada nota, además de no haber silencios.

Primero hay que alinear todas las notas que comienzan a sonar simultáneamente. Este proceso se conoce como cuantizado de las notas y consiste en ajustar su posición de inicio, de final o su duración de forma que se ajusten a una rejilla en la que el tamaño de cada celda de la rejilla está marcado por una duración de nota determinada. Cuanto mayor sea este tamaño, menor será la precisión, por lo que conviene tomar duraciones cortas como una semifusa por ejemplo. En la figura 7 se puede ver que ocurre cuando se cuantizan notas escogiendo una duración demasiado elevada. Las notas más largas preservan aproximadamente el tiempo de comienzo, pero aquellas más cortas que la duración escogida para cuantizar se desplazan tanto que el sonido se acaba modificando de forma notable.

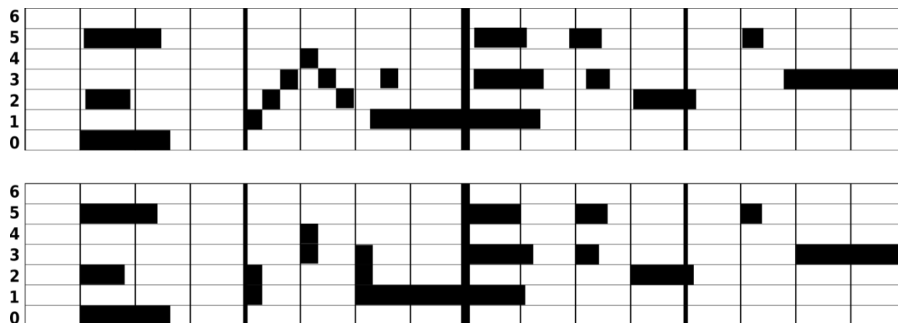

Figura 7: Un rollo de piano de un archivo MIDI antes (arriba) y después (abajo) de cuantizar sus notas al inicio

De esta forma se garantiza que todas las notas que percibimos que suenan a la vez pero no se están tocando exactamente al mismo instante queden alineadas entre sí para poder agruparlas correctamente.

Después, se toman todos los eventos note on que ocurren simultáneamente o lo que es lo mismo, se seleccionan todos los note on mientras el delta time entre todos ellos sea 0. En el momento que esto no ocurra, se toma el valor de las notas de todos ellos y se concatenan mediante cualquier carácter (un punto por ejemplo) que se empleará como separador más adelante a la hora de realizar el proceso inverso. Este proceso se

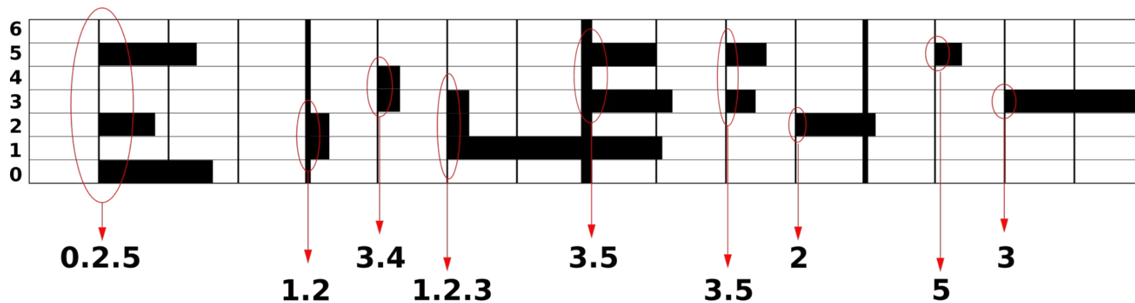

Figura 8: Trasformación de los eventos note on cada grupo de notas en una cadena de caracteres

vuelve a repetir con las notas siguientes para obtener un nuevo grupo en el que todas suenan simultáneamente hasta alcanzar el final de la pista MIDI. Cada vez que se genera un grupo se añade a una lista que una vez finalizado el proceso, se transforma en una cadena concatenando nuevamente todos sus elementos como un símbolo distinto al empleado antes, un guión en este caso. Este último paso solo es necesario para facilitar la visualización de la secuencia de notas como una cadena de texto y se puede omitir.

Para entrenar el modelo con los datos empleando esta representación, se añade cada uno de los grupos a un diccionario en el que se asocia a un número. Para que el proceso sea reversible, se crea un diccionario adicional en el que se realiza la asociación inversa. Se repite el proceso para todos los grupos de notas siguientes y finalmente se convierte la cadena de grupos de notas en una lista de números en la que cada número es el número asociado al grupo en la posición correspondiente empleando el diccionario de asociaciones grupo-número. Esta lista de números se convierte entonces en una matriz empleando una codificación one-hot de los números de la lista. Finalmente, se entrena como un problema de clasificación tratando de predecir a qué clase pertenece cada secuencia de vectores, siendo la clase a la que pertenece la posición en la que el vector correspondiente a la secuencia la posición en la que toma el valor 1.

2.2.3 Codificación de Performance RNN

Este tipo de representación trata de representar las duraciones de las notas y añadir silencios, además de añadir las dinámicas que tiene una persona a la hora de interpretar la música. Es la representación que emplea Performance RNN, una red neuronal recurrente del proyecto Magenta desarrollada con la finalidad de imitar el estilo de un músico real interpretando su instrumento.

En principio está pensada para funcionar con un único instrumento o pista, pero puede extenderse a múltiples instrumentos como es el caso de la codificación empleada por BandNet [4], una red que imita el estilo de los Beatles y usa y genera archivos MIDI multipista (una para cada instrumento). Esta red emplea una codificación similar a la

empleada por Performance RNN, pues usa un pseudoevento más (*next track*, para desplazarse entre pistas) y elimina el pseudoevento que modifica la velocidad *set velocity*.

Para transformar un MIDI a esta representación el primer paso es tomar todos los eventos *note on* y *note off*, y adicionalmente y como es nuestro caso, añadir mensajes de control para controlar la expresividad del instrumento. En este caso se ha tenido en cuenta el mensaje de control que controla el pedal de resonancia del piano por ser este pedal el más usado.

Después, hay que descomponer los mensajes correspondientes a estos eventos en pseudoeventos más simples, pues cada uno de estos eventos contiene mucha información como por ejemplo la intensidad con la que se ha pulsado la tecla o cuánto tiempo ha pasado desde que se ha pulsado última tecla además del valor de la propia nota. Para poder realizar esta descomposición se crean varios tipos de pseudoeventos:

- **note on:** Este evento indica el comienzo de una nota. Hay 128 de este tipo, uno por cada posible valor de nota.
- **note off:** Indica el final de una nota. También hay uno por cada posible valor de nota MIDI, esto es, 128 eventos.
- **set velocity:** Modifica la velocidad de las notas que le suceden. Es conveniente dividir el espectro de 128 velocidades posibles en 32 segmentos y asignar cada una de las 128 velocidades a un segmento empleando binning, puesto que generalmente no es necesaria tanta precisión.
- **time shift:** Realiza los desplazamientos en el tiempo. Se pueden realizar desplazamientos que van de duraciones entre 10 milisegundos hasta 1 segundo en incrementos de 10 milisegundos, habiendo por ello 100 mensajes de este tipo.

Como se ha mencionado antes, se ha añadido en este caso un evento para controlar el pedal de resonancia del piano al que se ha denominado simplemente **pedal,** y determina si el pedal se ha pisado o no, habiendo en consecuencia 2 mensajes de este tipo.

Para poder llevar a cabo esta descomposición en eventos más simples, es necesario avanzar evento a evento En estos ejemplos se van a emplear las velocidades sin emplear binning con la finalidad de simplificarlos un poco, a pesar de que a la hora de implementarlo se restringe el número de eventos de velocidad a 32 como se ha explicado antes.

Si el evento MIDI es un evento *note on* para la nota 57 con velocidad 98, se lanzan dos pseudoeventos: un pseudoevento *set velocity* primero con el valor 98 y un

pseudoevento *note on* con el valor 59 después. Si el siguiente evento es otro *note on* para la nota 74 pero mantiene la misma velocidad, solamente se lanza el pseudoevento *note on* con valor 59, pues la velocidad que se usa será la última que ha establecido mediante el evento *set velocity*. En el momento en el que un evento llega con un *delta time* mayor a cero, se lanza un evento *time shift*, transformando el *delta time* de ticks MIDI a milisegundos y a continuación calculando que pseudoevento permite llevar a cabo dicho desplazamiento en el tiempo con el menor error posible.

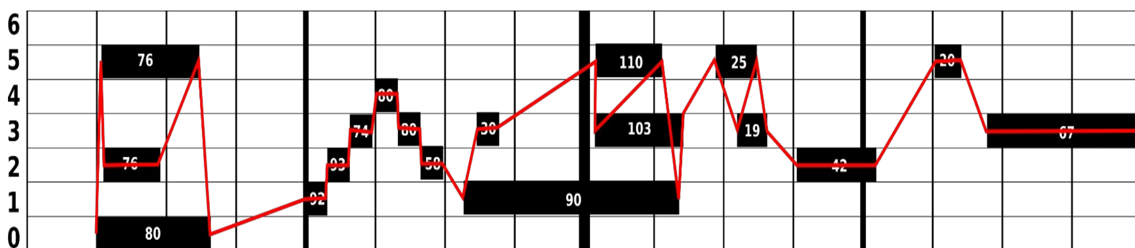

Figura 9: La línea roja marca el recorrido de los eventos MIDI necesario para obtener la secuencia de pseudoeventos en esta codificación. El número en blanco sobre cada nota es su velocidad. El comienzo de la secuencia de eventos resultante se encuentra en el texto 2

```
set_velocity 80, note_on 0, time_shift 0.1, set_velocity 76,
note_on 5, note_on 2, time_shift 0.8, note_off 2, time_shift
0.5, note_off 5, time_shift 0.1, note_off 0…
```

Texto 2: Primeros pseudoeventos de la secuencia obtenida a partir del rollo de la figura 9 empleando el codificación de Performance RNN

Como se puede ver, el número de eventos está acotado y no puede haber más de 128 eventos *note on*, 128 *note off*, 32 *set velocity* y 100 *time shift* además de los 2 de *pedal* que se han añadido, haciendo un total de 390 eventos posibles. Gracias a esto, es posible asignar un número a cada uno de ellos para después ser codificado empleando una codificación one-hot y como en el resto de representaciones, tratar de predecir la clase a la que pertenece cada una de las secuencias.

2.3 Señal de audio

A diferencia del resto de formatos que se presentan en este trabajo, los archivos que contienen una señal de audio en lugar de una serie de símbolos con significados en los que existe una cierta estructura no son más que una sucesión de puntos en los que no existe un conjunto de reglas predefinidas que permita combinarlos, la señal de audio se obtiene a partir de las vibraciones captadas por un micrófono. Como resultado, se obtiene una señal analógica continua en la que no existe contenido musicalmente semántico.

Para digitalizar la señal, se realiza un proceso de muestreo, tomando muestras de la onda en intervalos de tiempo regulares. La longitud de estos intervalos viene fijada por la frecuencia de muestreo. Cada uno de los valores de estas muestras se codifica empleando una técnica llamada PCM, dividiendo el espacio de posibles valores en intervalos de igual tamaño y cuantizando el valor de cada muestra al valor más cercano. El número de fragmentos viene determinado por la profundidad de bit. A mayor profundidad de bit, se puede dividir en un número mayor de intervalos, aumentando la fidelidad del sonido.

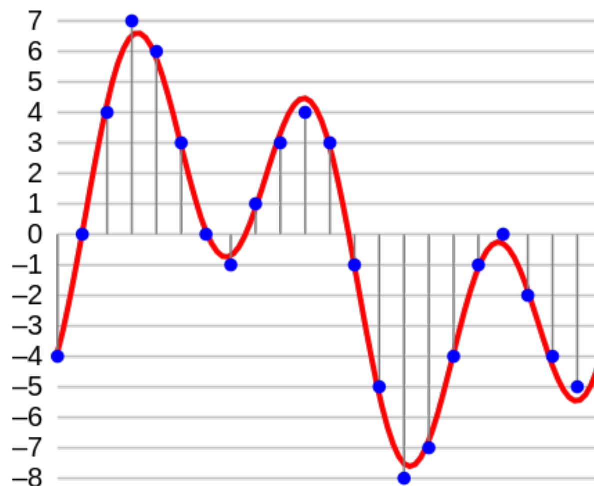

Figura 10: Codificación de una señal analógica (en rojo) tomando muestras a intervalos regulares de tiempo empleando una profundidad de 4 bits. Nótese como cada muestra toma el valor más cercano a la señal dentro de todos los valores posibles.[5]

Debido a la validez del teorema de muestreo de Nyquist-Shannon, es posible recuperar la señal original sin pérdidas a partir de dichas muestras siempre y cuando la frecuencia de muestreo sea al menos del doble de la frecuencia máxima. Como las señales de audio son señales en banda limitada puesto que los humanos no podemos escuchar frecuencias inferiores a 20 Hz y superiores a 20 kHz, basta con tomar muestras con una frecuencia de 40 kHz para que no se puedan percibir las pérdidas. Es el caso de los CDs de audio que emplean una frecuencia de muestreo de 44,1 kHz y una profundidad de bit de 16 bits, pudiendo representar hasta 65.536 valores.

Su principal problema es la gran cantidad de información y su poco valor semántico. Debido a la enorme cantidad de muestras por segundo en un archivo de este tipo, son necesarios una gran cantidad de recursos y de tiempo.

Para solventar este problema, se suele reducir la frecuencia de muestreo y la profundidad de bit, a costa de perder calidad en la señal. La frecuencia empleada suele ser de 16kHz y la profundidad de bit de 8 bits.

5 Imagen por Aquegg bajo licencia CC BY-SA 3.0

A la hora de emplear este tipo de representación, se suele tratar el problema como un problema de clasificación, en el que dada una sucesión de valores para varias muestras, se debe obtener cual es el número que toma la amplitud de entre todos los posibles valores marcados por la profundidad de bit. Supongamos por ejemplo, que se quiere predecir el valor de una muestra dadas las 100 muestras anteriores con una profundidad de bit n . Para ello, se transforma el valor de la amplitud señal que se encuentra en el intervalo $[-1,1]$ para que pase al intervalo $[0,1]$. Después se multiplica por 2^n-1 para tenerlo en el intervalo $[0,2^n-1]$. Finalmente se redondea para obtener un número entero como resultado. Este número corresponde al número de la clase a la que pertenece la sucesión de muestras entregada a la entrada y como en el resto de casos, se codificar como un vector one-hot. Para deshacer el proceso y obtener el valor de la muestra a partir de la clase, se despeja la ecuación para la variable x .

$$q(x,n)=\frac{(2^n-1)(x+1)}{2}$$

Figura 11: Fórmula para calcular la clase a la que pertenece una muestra dado el valor de la muestra x y la profundidad de bit n

$$q^{-1}(x,n)=\frac{2x}{2^n-1}-1$$

Figura 12: Fórmula para calcular la el valor de la muestra a partir de la clase x y la profundidad de bit n

3 Redes neuronales recurrentes

Las redes neuronales convencionales no son el tipo de red adecuado para tratar con secuencias, puesto que la longitud de todas las secuencias a tratar no es la misma y estas redes carecen de mecanismos que permitan almacenar información sobre los estados anteriores de la secuencia. Es por esta razón por la que es necesario emplear otro tipo de redes: las redes neuronales recurrentes.

Este tipo de redes permiten tratar con secuencias de longitudes distintas y se pueden emplear para resolver problemas de clasificación y regresión del mismo modo que las redes convencionales. Para poder manejar secuencias de longitud variable, las redes de este tipo se desenrollan para poder recibir en cada una de las etapas el valor de la secuencia en su respectivo instante, y comunican la información sobre los valores anteriores de la secuencia entre cada una de las etapas. De esta forma, la longitud de la secuencia pasa a ser irrelevante a la vez que en cada uno de los instantes se tiene información sobre los estados anteriores, añadiendo contexto. Adicionalmente,

permiten tener también un número indeterminado de salidas, pudiendo así obtener también secuencias de longitudes sin determinar a la salida de la red.

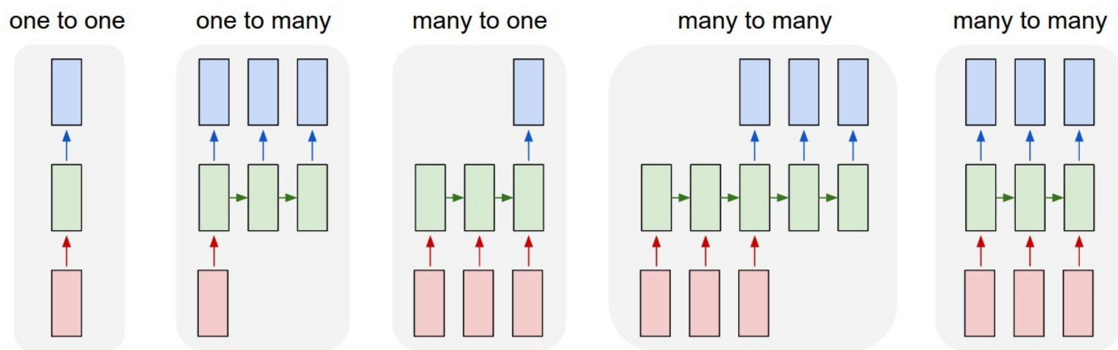

Figura 13: Posibles configuraciones de una red recurrente frente a una red convencional[6]

Estas diferencias se pueden ver mejor en la figura 13, donde se muestran las posibles configuraciones de una red neuronal recurrente junto a la única configuración posible de las redes convencionales. En la imagen los rectángulos rojos representan los vectores de entrada, los verdes el estado de la red y los azules los vectores de salida. Si se mira la imagen de izquierda a derecha, la primera configuración es *one to one*, la que usan las redes convencionales, que solo permite un número limitado de valores tanto a la entrada como a la salida. El resto de configuraciones corresponden a las posibles formas de organizar una red neuronal recurrente, cada una de ellas con distintas aplicaciones. La segunda *(one to many),* permite que la entrada sea de tamaño fijo pero la salida pueda tener una longitud sin determinar. Esta configuración se puede emplear por ejemplo para la descripción de imágenes, recibiendo una imagen a la entrada y produciendo una frase describiendo con lenguaje natural los elementos que aparecen en ella. Las redes neuronales recurrentes también se pueden usar para la clasificación de secuencias *(many to one)*, permitiendo por ejemplo que dada una canción, sea capaz de decirnos en que tono se encuentra o quien es su autor, o para la traducción automática (many to many) en la que se recibe una frase formada por un número no determinado de palabras en un idioma y se transforma en la misma frase en otro idioma, también de longitud indefinida. La otra configuración de *many to many* se emplea cuando existe una correspondencia uno a uno entre cada uno de los elementos de la entrada y de la salida, como puede ser un vídeo en el que se desea clasificar cada uno de los fotogramas.

Una vez solventado el problema de las longitudes de las secuencias, es necesario emplear algún tipo de elemento capaz de retener la información sobre estados anteriores en la secuencia para poder mejorar la predicción, ya que esta capacidad no está disponible en las redes neuronales convencionales. Con el paso de los años, se

6 Fuente: https://karpathy.github.io/2015/05/21/rnn-effectiveness/

fueron desarrollando distintos tipos de redes recurrentes con la capacidad de manejar secuencias de longitud indeterminada, pero todas ellas debían superar el mismo obstáculo: El desvanecimiento de gradiente. Este problema ocurre en todas las redes neuronales en las que se usan métodos de optimización basados en el descenso por gradiente y la retropropagación y es causado al actualizar los pesos de la red y recibir una modificación proporcional a la derivada parcial de la función de error con respecto al peso en cada iteración del entrenamiento. Los gradientes de las funciones de activación suelen ser valores pequeños en el intervalo $(0,1)$ y al multiplicar todos los valores de cada capa en el proceso de retropropagación se acaban alcanzando valores mucho más pequeños en las primeras capas de la red, ralentizando el entrenamiento de estas capas, mientras que en las capas más profundas el valor es mayor lo que conlleva a que los pesos se actualicen más rápido.

El desvanecimiento se manifiesta de forma especial en las redes recurrentes porque su estructura recurrente puede ser "desenrollada" en múltiples capas. Como consecuencia de este proceso de desenrollado, al propagar los gradientes en cada capa, la actualización de los pesos de las capas más cercanas al inicio se realiza de forma mucho más lenta favoreciendo que los primeros elementos en la secuencia no se tengan en cuenta a la hora de realizar la predicción.

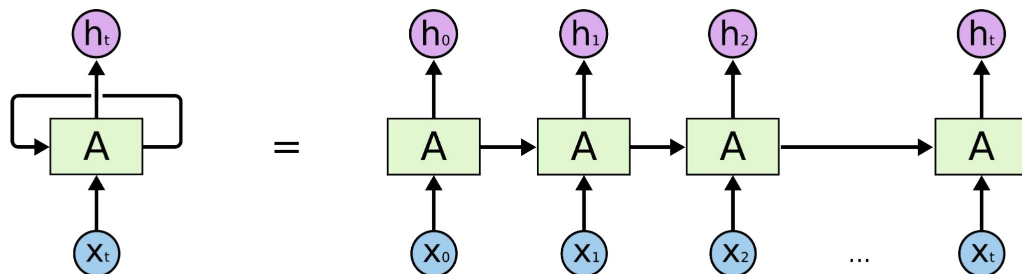

Figura 14: Red recurrente "desenrollada"[7]

Una de las formas más exitosas de tratar de resolverlo es el empleo de células LSTM o *long short-term memory.* Las células LSTM fueron propuestas en 1997 por Sepp Hochreiter y Jürgen Schmidhuber [5], que resuelven el problema de forma parcial al introducir una serie de puertas que permiten modificar el estado o preservarlo tal como está, permitiendo así que se pueda almacenar información sobre instantes anteriores durante mayor tiempo.

En 2014 fue propuesta [6] otro tipo de células más simples que las células LSTM y con resultados similares a estas: las células GRU o *gated recurrent unit.* Este tipo de células tienen una puerta menos que las LSTM, reduciendo el número de parámetros . Aun así, las células LSTM han demostrado ser mucho más eficaces para la resolución de determinados problemas.

7 Fuente: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

3.1 LSTM

Una célula LSTM en su propuesta original está formada por 2 puertas: Una puerta de entrada o *input gate* y una de salida u *output gate*. En 1999 se propuso añadir una puerta adicional a la que se le dio el nombre de *forget gate,* una puerta que permite eliminar los valores del estado que ya no se consideran necesarios para la predicción. Estas tres puertas se encargan de regular la información que entra, sale y se queda dentro de la célula.

Cada una de las puertas funciona como una red neuronal convencional: Recibe unos datos de entrada, los multiplica por sus pesos (denotados en las ecuaciones con la letra W) y les suma el *bias* (en las ecuaciones se emplea la letra b) y a continuación se les aplica una función de activación. El resultado de esta función de activación regula que información pasa por la puerta correspondiente.

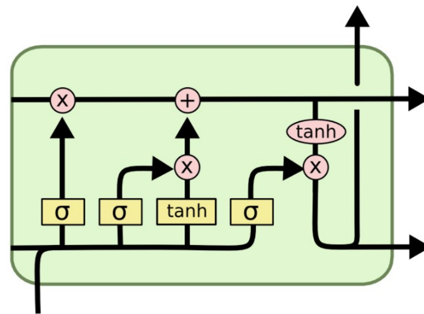

Figura 15: Diagrama de una célula LSTM[8]

Una célula LSTM está formada por:

- El estado de la célula C
- El estado oculto de la célula h
- Una entrada x
- Una puerta *input* con sus parámetros W_i y b_i
- Una puerta *forget* con los parámetros W_f y b_f
- Una puerta *output* con los parámetros W_o y b_o

La entrada x_t recibe los datos en el punto de la secuencia que le corresponde, mientras que las otras dos entradas reciben el estado de la célula C y el estado oculto de la célula inmediatamente anterior h_{t-1} en la secuencia. El estado de la célula actúa de forma similar a una memoria en la que se almacena la información

8 Fuente: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

sobre instantes anteriores en la secuencia y su valor se modifica mediante las puertas *input* y *forget*. Para ello, primero se combinan el estado oculto con la entrada de la célula concatenando las columnas de las matrices que reciben como entrada y se les multiplican los pesos W_f y se añade el bias b_f para a continuación aplicar la función sigmoide como activación.

$$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f)$$

Figura 16: Ecuación para la puerta forget

El resultado de esta función se encuentra en el intervalo $[0,1]$, por lo que al multiplicar elemento a elemento su salida con el estado de la célula C_{t-1} (como ocurre en la ecuación en la figura 19) se decide qué valores se conservan y cuáles son descartados u olvidados (de ahí el nombre de esta puerta). Después, se repite este proceso de forma similar para la puerta *input*. Esta puerta aplica la función sigmoide y la tangente hiperbólica por separado a la combinación del estado oculto y la entrada para después multiplicar elemento a elemento sus salidas. De esta forma se seleccionan qué valores se van a añadir al estado a la vez que se calculan qué valores se van a añadir.

$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i)$$

Figura 17: Ecuación para la puerta input

$$C_t = \tanh(W_C[h_{t-1}, x_t] + b_C)$$

Figura 18: Ecuación para calcular los valores que se añadirán al estado

Para actualizar el estado de la célula, se suman el estado y la salida de la puerta *input* elemento a elemento.

$$C_t = f_t * C_{t-1} + i_t * C_t$$

Figura 19: Ecuación para actualizar el estado de la célula

Como resultado de esta operación se ha obtenido el siguiente estado C_t , pero no el estado oculto h_t . Para ello, se calcula el valor de salida de la célula o_t aplicando la función sigmoide sobre la suma del producto de la matriz de pesos W_o la combinación del estado oculto h_{t-1} y la entrada x_t con el *bias* b_o . Para calcular el nuevo estado oculto h_t se multiplica elemento a elemento la salida de la célula o_t por el resultado de aplicar la tangente hiperbólica sobre el estado C_t

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o)$$

Figura 20: Ecuación para calcular la salida

$$h_t = o_t * \tanh(C_t)$$

Figura 21: Ecuación para calcular el estado oculto

3.2 GRU

Una célula GRU funciona de forma parecida a una LSTM. Este tipo de célula elimina el estado de la célula y emplea el estado oculto para almacenar y transferir la información de instantes anteriores. Además, se deshace de las puertas *forget* e *input* e introduce un nuevo tipo de puerta que combina la funcionalidad de ambas, la puerta *update gate*, que se encarga de decidir qué valores de la entrada se añaden al estado de la célula y cuáles no. La otra puerta de la que disponen las células GRU es la puerta *reset*, que se encarga de descartar los valores que no se consideran importantes.

La puerta *update* tiene asociados los parámetros W_z y b_z mientras que los parámetros para la puerta *reset* son W_r y b_r. Como ya se ha mencionado en el párrafo anterior, la célula GRU dispone únicamente de un estado oculto cuyo valor valor está denotado por h .

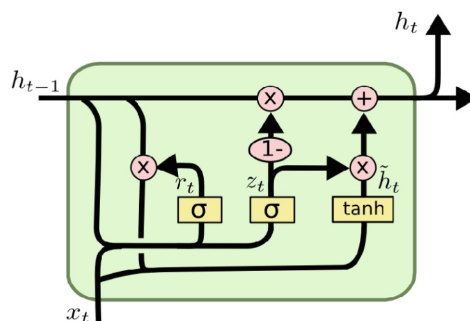

Figura 22: Diagrama de una célula GRU[9]

Del mismo modo que una célula LSTM, recibe el estado oculto anterior h_{t-1} y la entrada del instante correspondiente x_t y las combina. Esta combinación pasa a la puerta *reset*, donde se calcula r_t que toma valores entre 0 y 1, y se multiplica por los datos de entrada, decidiendo así si se añaden al estado o se descartan (si son multiplicados por valores cercanos a 1 o a 0 respectivamente).

$$r_t = \sigma(W_r[h_{t-1}, x_t] + b_r)$$

Figura 23: Ecuación para la puerta reset

A continuación, se repite el proceso para la puerta *update*, que calcula z_t de la misma forma que se ha calculado r_t en la puerta *reset*. Este valor se emplea para indicar los valores que van a ser ignorados en el estado y calcular el nuevo estado oculto.

9 Fuente: https://technopremium.com/blog/rnn-talking-about-gated-recurrent-unit/

$$z_t = \sigma(W_z[h_{t-1}, x_t] + b_z)$$

Figura 24: Ecuación para la puerta update

Para actualizar el estado de la célula en este caso, primero se calcula los valores que se van a añadir $\hat{h}_t$ y después se calcula una suma ponderada de los valores anteriores del estado h_{t-1} con los valores en $\hat{h}_t$.

$$\hat{h}_t = \tanh(W[r_t * h_{t-1}, x_t] + b)$$

Figura 25: Ecuación para calcular los valores que se añadirán al estado oculto

$$h_t = (1 - z_t) * h_{t-1} + z_t * \hat{h}_t$$

Figura 26: Fórmula para calcular el nuevo estado oculto

3.3 Generación de secuencias mediante redes neuronales recurrentes

Para generar secuencias empleando redes recurrentes es necesario recorrer cada secuencia con una ventana de un tamaño previamente fijado. Los valores dentro de cada una de estas ventanas se pasan a la entrada de la red que debe predecir el siguiente valor de la secuencia.

Esta predicción se realiza típicamente como un problema de clasificación aunque se podría tratar como un problema de regresión si la secuencia está formada por valores continuos, como en el caso de la predicción de precios por ejemplo.

Una de las aplicaciones habituales de este tipo de redes para generar secuencias es la generación de texto. Para llevar a cabo esta tarea, se toma un conjunto de textos, que pueden ser tratados como secuencias de caracteres o palabras. Cada secuencia se recorre con una ventana de tamaño n , y se trata de predecir a qué clase pertenece, siendo esta clase el siguiente carácter o palabra en la secuencia. Después de predecir el siguiente valor, se concatena al final de la secuencia de entrada y se desplaza la ventana para volver a predecir el siguiente valor tal y como se ve en la figura 27. Este proceso en principio se puede repetir indefinidamente, pero se suele fijar previamente una longitud máxima de secuencia o se continúa generando la secuencia hasta que aparezca un carácter de fin de secuencia.

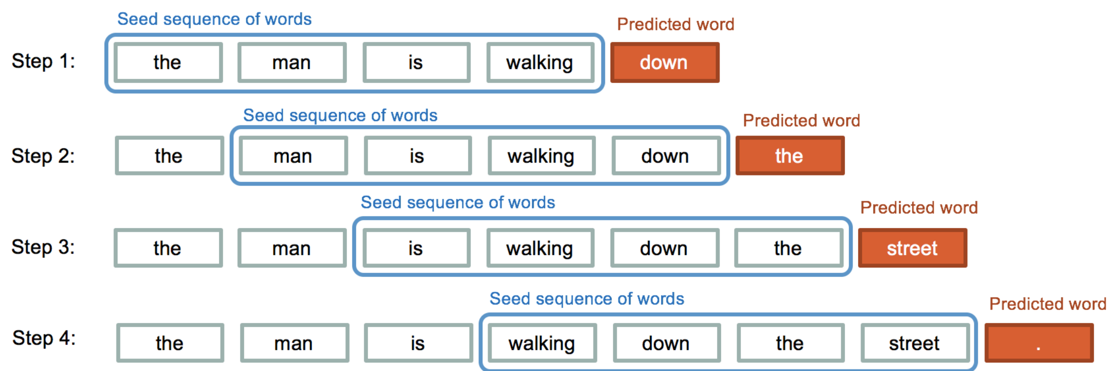

Figura 27: Generación de texto palabra por palabra[10]

4 Redes neuronales profundas para la generación de música

Tras estudiar las diferentes alternativas para la codificación de música, en esta sección vamos a desarrollar algunas arquitecturas basadas en aprendizaje profundo para la generación automática de música.

Estas representaciones se han evaluado construyendo distintos modelos con *Keras* en su versión incluida dentro de *Tensorflow*. Los modelos se han entrenado en *Google Colab*, una herramienta de Google que permite ejecutar notebooks en sus GPU o TPU de forma gratuita durante un tiempo limitado. Se han usado además las librerías *matplotlib* para la generación de gráficos, *numpy* para algunos cálculos en el preprocesamiento, *librosa* para el procesamiento de audio, *music21* para extraer los tokens de los ficheros ABC y *mido* para el procesamiento de archivos MIDI.

Los datasets empleados para entrenar estos modelos son totalmente públicos y en cada uno de los experimentos se especifica su origen. En algunos casos se han aplicado técnicas de aumentación de datos, puesto que el dataset original tiene muy pocos ejemplos.

Con la finalidad de poder escuchar los los resultados de cada arquitectura, toda la música generada empleando estos modelos está disponible en un repositorio[11] de GitHub.

La longitud de todas las piezas musicales es variable, por lo tanto se debe de escoger un tamaño de secuencia y fijarlo para recorrer cada pieza empleando una ventana de dicho tamaño. En cada uno de los experimentos se utiliza un tamaño de secuencia distinto en función de las características de la codificación.

10 Fuente: https://iq.opengenus.org/text-generation-lstm/

11 https://github.com/ibonn/musica_tfg

Estas redes deberán predecir además el siguiente símbolo en la secuencia, un símbolo que se encuentra dentro de un conjunto finito de símbolos, por lo que se trata de un problema de clasificación multiclase en el que el número de clases está determinado por el número de símbolos en el conjunto. En todos los casos el número de clases exactamente el número total de símbolos en el conjunto de entrenamiento.

4.1 Generación carácter a carácter empleando notación ABC

Para generar música con esta representación, se ha empleado una red con dos capas, una capa con 128 células LSTM y una capa de salida densamente conectada con tantas neuronas como caracteres/tokens se hayan encontrado en todas las canciones. Como es un problema de clasificación en el que cada ejemplo solo puede pertenecer a una clase, se ha escogido softmax como la función de activación de la última capa.

El dataset empleado para esta codificación es la versión en notación ABC de *Nottingham Music Database, un dataset* formado por 1065 canciones, la mayoría de ellas canciones folk, aunque tambien se incluyen valses y villancicos y canciones navideñas. Cada una de las canciones se ha transformado de la forma descrita en 2.1.1 para después dividirla en secuencias de 40 caracteres y tratar de predecir el siguiente carácter de cada una de estas subsecuencias. se ha entrenado durante 100 epochs empleando *rmsprop* como algoritmo de optimización.

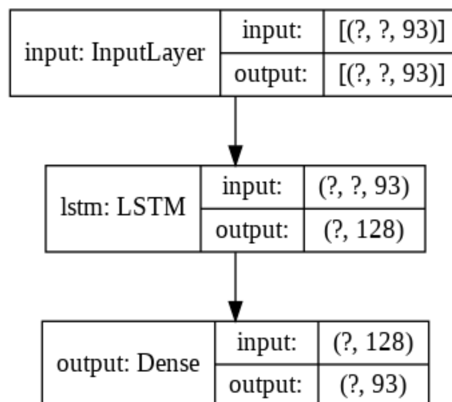

Figura 28: Red empleada para la generación carácter a carácter

```
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 128)               113664
_________________________________________________________________
output (Dense)               (None, 93)                11997
=================================================================
Total params: 125,661
Trainable params: 125,661
Non-trainable params: 0
_________________________________________________________________
```

Figura 29: .summary() de la red

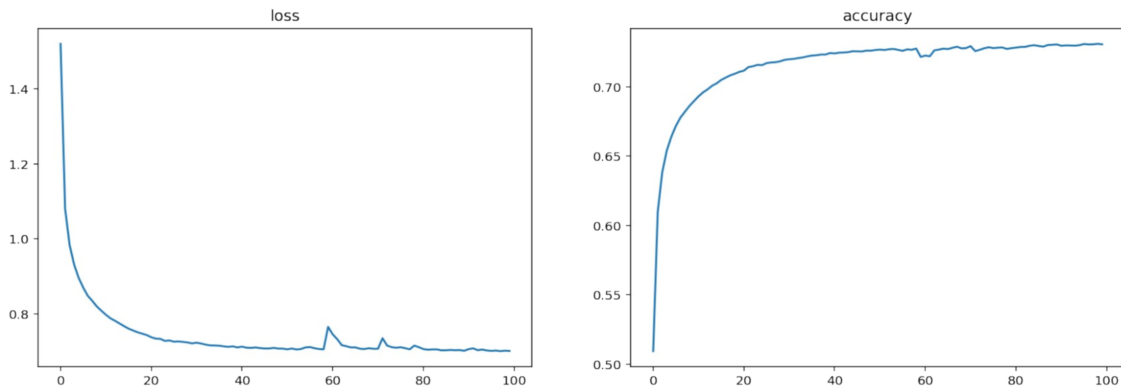

Figura 30: loss y accuracy en train para la generación carácter a carácter

Para generar los resultados, se emplea un factor llamado temperatura que introduce una mayor variabilidad y permite controlar el resultado de la salida. Para valores de temperatura mayores, mayor es la aleatoriedad de los resultados mientras que si la temperatura es más baja, las secuencias resultantes tienden a ser más repetitivas. Si la temperatura es 1, no se realiza ningún ajuste en la predicción. Se puede entender como cuánto se arriesga cuando se hace la predicción, siendo una predicción conservadora si la temperatura es baja y más arriesgada cuando la temperatura es alta. La predicción en función a la temperatura y las probabilidades de salida se lleva a cabo mediante la función *sample* en el código, como se puede ver en la figura 31.

```
# Se selecciona una semilla
seed = 'X:'

# Generar distintas canciones con temperaturas diferentes
for temp in [0.1, 0.2, 0.5, 0.8, 1.0, 1.2, 2]:

    generated = seed
    sentence = seed

    # Generar la canción
    pbar = ProgressBar(prefix='Generando muestra con temperatura {}...'.format(temp))
    for i in pbar(range(400)):
        x_pred = char2num(generated).reshape(1, -1, n_chars)
        preds = model.predict(x_pred, verbose=0)[0]
        next_index = sample(preds, temp)
        next_char = indices_char[next_index]
        sentence = sentence[1:] + next_char
        generated += next_char

    # Guardar el archivo en un fichero
    fn_abc = '/content/drive/My Drive/TFG/gen/gen_abc_char_{}.abc'.format(temp)
    fn_mid = '/content/drive/My Drive/TFG/gen/gen_abc_char_{}.mid'.format(temp)
    with open(fn_abc, 'w') as f:
      f.write(generated)

    # Convertir a MIDI
    result = os.popen('abc2midi "{}" -o "{}"'.format(fn_abc, fn_mid)).read()
    print('Muestra generada. Resultado de abc2midi: {}'.format(result))
    print()
```

Figura 31: Código empleado para generar las secuencias de caracteres

Como consecuencia se puede ver que la música que se genera no es aceptada por algunos intérpretes de notación ABC si la temperatura es más alta. Es el caso de las temperaturas superiores a 0.5, que han dado lugar a secuencias que no son consideradas válidas por abcmidi, un programa que convierte las piezas en notación ABC a MIDI para poder ser escuchadas. Aun así, otros intérpretes dan por válidas estas secuencias.

La generación se lleva a cabo entregando un valor inicial como semilla. En este caso, al haber incluido el campo X en todas las piezas de entrenamiento y ser este obligatorio, se usa la semilla X: y se trata de completar a partir de ahí. Si todo va bien, esto nos debería dar como resultado la pieza completa pudiendo incluso incluir autor y título aunque no los necesitemos.

A pesar de haber incluido todas las cabeceras en el entrenamiento, hay algunas que no aparecen en ninguna de las muestras generadas, como por ejemplo el título y el autor y otras muchas aparecen mal formadas Es el caso del ejemplo en el texto 3, que no contiene cabeceras de autor ni título, pero sí que contiene la cabecera de compás (M), duración por defecto (L) y tono (K). La cabecera correspondiente a el número de la pieza se ha completado de forma inválida, pues debería ser un número y es una cadena de texto que no tiene ningún sentido. Todas ellas a pesar de ser válidas y ser convertidas a MIDI sin ningún problema contienen caracteres inválidos que son

ignorados, como es el caso de la pieza generada con temperatura 0,1 (en el texto 3) que se convierte a MIDI correctamente pero muestra algunos errores durante la conversión visibles en el texto 4.

```
X:PC\7
3/3/G6CCCCCCCCCCCCorrrrrr Dam19m1985, via PR
M:4/4
L:1/4
K:G
|:d/2c/2|"G"BG GA|"G"Bd d2|"Am"c2 -"D7"d"3E/2 -
DG|"G"B4\
|"D7"^c2D|"G"G4-|B4\
|"G"B2 A2|"G"B4-|"A7"^D2g|"D"a4-|"D6/2
P

AG/2A/2 |"Gm"d2|"Gm"gf/2g/2 -g/2a/2b|"F"a2
=f=c|"Cm"=c=c=c|^C/2^G/2^A/2 ^G/2=C==^=c/2^G/4^G/4
^c/4^G/4A/4|(3c/4=B/4^d/4^g/4 (3"F7"=f/4^d/4^g/4
^f/4^g/4|g/4f/4:|/2a/4f/4|(3g/4g/4f/4^g/4|(3c3/4|
d/4f/4|"C"==f/4g/4 g/
```

Texto 3: Pieza generada con temperatura 0,1

```
Error in line-char 0-0 : Missing Number
Warning in line-char 2-0 : Ignoring text: 3/3/G6CCCCCCCCCCCCorrrrrr
Dam19m1985, via PR
Error in line-char 6-52 : Guitar chord name not properly closed
Error in line-char 6-52 : Unrecognized chord name "4\"
(Hint: use %MIDI chordname to define it. eg %MIDI chordname sus4 0 4
7).
Error in line-char 8-33 : Guitar chord name not properly closed
Error in line-char 8-33 :  / must be followed by A-G or a-g in gchord
Warning in line-char 9-0 : Ignoring reserved character P
Error in line-char 6-0 : Could not find note to be tied
Error in line-char 7-0 : Could not find note to be tied
Error in line-char 8-0 : Could not find note to be tied
Error in line-char 8-0 : Could not find note to be tied
writing MIDI file /content/drive/My Drive/TFG/gen/gen_abc_char_0.1.mid
Error in line-char 10-0 : Missing :| at end of tune
Warning in line-char 7-9 : Track 1 Bar 4 has 3 units instead of 4
Warning in line-char 8-25 : Track 1 Bar 9 has 3 units instead of 4
Error in line-char 10-0 : Missing :| at end of tune
Error in line-char 10-0 : Missing :| at end of tune
```

Texto 4: Salida del comando abc2midi para la pieza del texto 3

Los casos en los que no se ha podido convertir a MIDI tienen todos el mismo fallo: No contienen una cabecera de tono (K) o la que tienen no es válida. Si se usa un conversor menos estricto[12] las piezas se deberían escuchar sin ningún problema asumiendo que

12 En este caso se ha usado el conversor online https://clivew.com/abc.php

se encuentran en la tonalidad de do mayor y pudiendo haber alguna disonancia. Aún así, este editor evalúa el código de forma distinta a como lo hace abcmidi y si lo que se quiere es comparar los resultados con diferentes temperaturas lo mejor es añadir una cabecera de tonalidad a todas las piezas que no la tienen.

Al hacer esto, se puede ver que por lo general todas las que no tenían una cabecera de tonalidad son mucho más caóticas, con sucesiones rápidas de notas y sin un ritmo aparente a diferencia de las que se han generado con la cabecera de tono que sí que mantienen aproximadamente el ritmo.

4.2 Generación token a token con notación ABC

En el caso de la generación token a token, se ha modificado la arquitectura por una con más capas debido al mayor número de atributos y clases de este tipo de codificación. También se ha acortado la longitud de las secuencias de entrada a 10 tokens. Las piezas en notación ABC son las mismas que se han empleado para entrenar el modelo de la sección 4.1.

El preprocesamiento es el mismo que en el caso de la generación carácter a carácter, a diferencia de que en este caso se emplea un parser que se encarga de transformar cada uno de los archivos ABC en una secuencia de tokens. Para realizar este proceso, se ha empleado la clase ABCHandler de la librería *music21.* Esta clase contiene un método `tokenize` que recibe una cadena de texto y devuelve una lista de tokens ABC . Los tokens pueden ser de la clase ABCBar, que representa todos los tipos de barras de compás, ABCMetadata en el que se incluyen todos los campos posibles que pueden aparecer en cabeceras o en medio de la pieza y ABCNote, que incluye todas las notas y acordes posibles junto con sus duraciones.

```
<music21.abcFormat.ABCNote '"D7"f'>
<music21.abcFormat.ABCMetadata 'T:The American Dwarf'>
<music21.abcFormat.ABCBar ':|'>
```

Texto 5: Ejemplos de los tres tipos posibles de token en music21

Es necesario saber esto porque la música generada con este método requiere añadir saltos de línea después de cada cabecera para que el parser sea capaz de reconocerlo como un fichero válido. Si no se hace esto, las cabeceras se combinan en una misma línea junto con la música, dando como resultado código ABC inválido. Es por ello que en los diccionarios en los que se asocia un número con cada token se guardará el valor del token, pero en un diccionario separado se guardará un par clave valor asociando el valor de cada token con su tipo. Así, cada vez que se genere un token de clase ABCMetadata se deberá añadir un salto de línea después.

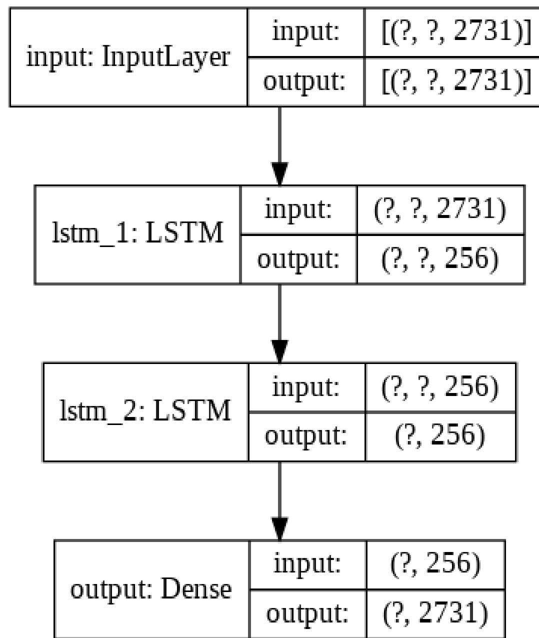

Figura 32: Arquitectura empleada para la generación token a token

```
Layer (type)                 Output Shape              Param #
=================================================================
lstm_1 (LSTM)                (None, None, 256)         3059712
_________________________________________________________________
lstm_2 (LSTM)                (None, 256)               525312
_________________________________________________________________
output (Dense)               (None, 2731)              701867
=================================================================
Total params: 4,286,891
Trainable params: 4,286,891
Non-trainable params: 0
_________________________________________________________________
```

Figura 33: .summary() para la red en el caso de generación token a token

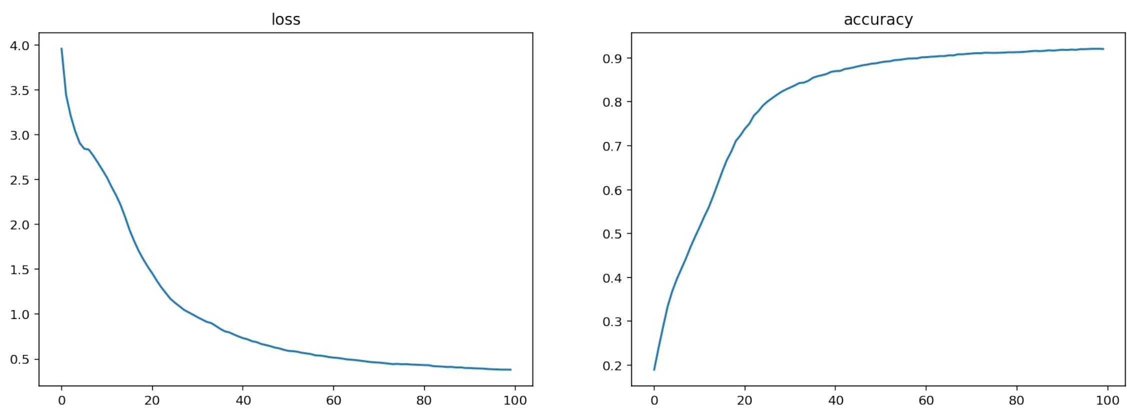

Figura 34: Loss y accuracy para la generación token a token

Una vez preparado todo, se generan secuencias empleando la misma técnica de la temperatura. Se selecciona como semilla la secuencia de 10 primeros tokens de una canción aleatoria del dataset original. En este caso, se han podido generar archivos MIDI para todos los ficheros ABC generados sin importar la temperatura, a pesar de que todos ellos contienen errores. En este caso se emplea como semilla los 10 primeros tokens de una pieza seleccionada aleatoriamente de todas las disponibles. Todos los ejemplos generados son convertidos a MIDI satisfactoriamente aunque todos muestran errores principalmente de compases que duran más o menos de lo que deberían o falta de barras de fin de repetición de sección. Esto lleva a que al escuchar el resultado se tenga la sensación de que se produce un corte y un salto repentino y que haya secciones que se deberían repetir que no se repiten.

```
song_len = 400
ws = 10

# Se selecciona una semilla
song = encoded_songs[np.random.randint(0, len(encoded_songs))]
seed = song[:ws, :].reshape(1, -1, n_chars)

# Generar distintas canciones con temperaturas diferentes
for temp in [0.1, 0.2, 0.5, 0.8, 1.0, 1.2, 2]:

    song = np.zeros((1, song_len, n_chars))
    song[:, :ws, :] = seed

    # Generar la canción
    pbar = ProgressBar(prefix='Generando muestra con temperatura {}...'.format(temp))
    for i in pbar(range(song_len - ws)):
        preds = model.predict(song[:, i:i+ws, :])
        next_index = sample(preds, temp)
        song[:, i+ws, next_index] = 1

    # Guardar el archivo en un fichero
    fn_abc = '/content/drive/My Drive/TFG/gen/gen_abc_token_{}.abc'.format(temp)
    fn_mid = '/content/drive/My Drive/TFG/gen/gen_abc_token_{}.mid'.format(temp)

    generated = 'X: 1\n{}'.format(num2char(song[0]))
    with open(fn_abc, 'w') as f:
      f.write(generated)

    # Convertir a MIDI
    result = os.popen('abc2midi "{}" -o "{}"'.format(fn_abc, fn_mid)).read()
    print('Muestra generada. Resultado de abc2midi: {}'.format(result))
    print()
```

Figura 35: Código para predecir el siguiente token

El resultado obtenido empleando este método es mucho más agradable que el obtenido generando carácter a carácter aunque parece que el modelo acaba sobreajustando y hay secciones que pueden ser copiadas directamente del conjunto de entrenamiento.

```
Warning in line-char 5-355 : Ignoring reserved character P
Warning in line-char 6-15 : Different length notes in tuple
Warning in line-char 6-16 : Different length notes in tuple
Warning in line-char 6-87 : Ignoring reserved character P
Warning in line-char 7-205 : Ignoring reserved character P
writing MIDI file /content/drive/My Drive/TFG/gen/gen_abc_token_0.5.mid
Warning in line-char 5-36 : Track 1 Bar 3 has 11/2 units instead of 6
Warning in line-char 5-160 : Track 1 Bar 13 has 5 units instead of 6
Warning in line-char 5-354 : Track 1 Bar 28 has 9 units instead of 6
Warning in line-char 6-25 : Track 1 Bar 29 has 32/3 units instead of 6
Warning in line-char 6-35 : Track 1 Bar 30 has 5 units instead of 6
Warning in line-char 5-36 : Track 1 Bar 3 has 11/2 units instead of 6 in
repeat
Warning in line-char 5-160 : Track 1 Bar 13 has 5 units instead of 6 in
repeat
Warning in line-char 7-6 : Track 1 Bar 15 has 2 units instead of 6 in repeat
Warning in line-char 7-148 : Track 1 Bar 28 has 7 units instead of 6 in
repeat
Warning in line-char 8-1 : Track 1 Bar 33 has 7 units instead of 6 in repeat
Warning in line-char 8-99 : Track 1 Bar 41 has 9/2 units instead of 6 in
repeat
```

Texto 6: Salida de abc2midi para el archivo con temperatura 0,8

4.3 Generación MIDI simple

Para este caso se ha empleado una red con una complejidad intermedia entre las empleadas para la generación carácter a carácter y token a token en notación ABC. El conjunto de datos empleado en este caso son ficheros MIDI provenientes de piano-midi.de[13], las 48 piezas de Chopin disponibles en la página, concretamente. El modelo se ha entrenado durante 100 épocas y se han empleado secuencias de 10 eventos note on.

13 http://www.piano-midi.de/

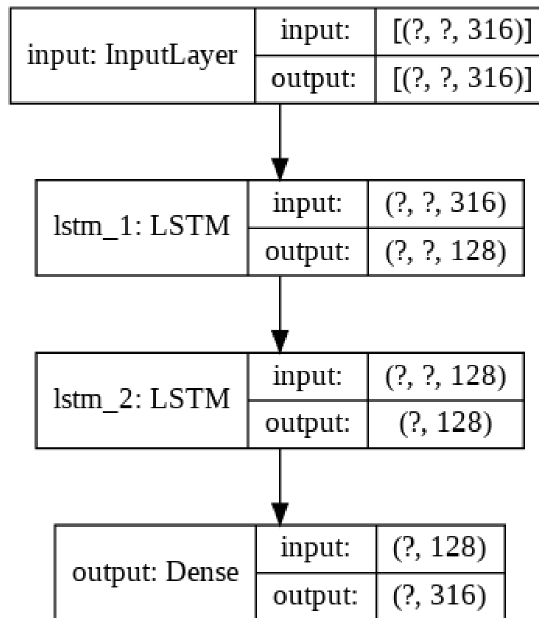

Figura 36: Red empleada para la generación MIDI con eventos note on

```
Layer (type)                 Output Shape              Param #
=================================================================
lstm_1 (LSTM)                (None, None, 128)         227840
_________________________________________________________________
lstm_2 (LSTM)                (None, 128)               131584
_________________________________________________________________
output (Dense)               (None, 316)               40764
=================================================================
Total params: 400,188
Trainable params: 400,188
Non-trainable params: 0
_________________________________________________________________
```

Figura 37: .summary() de la red

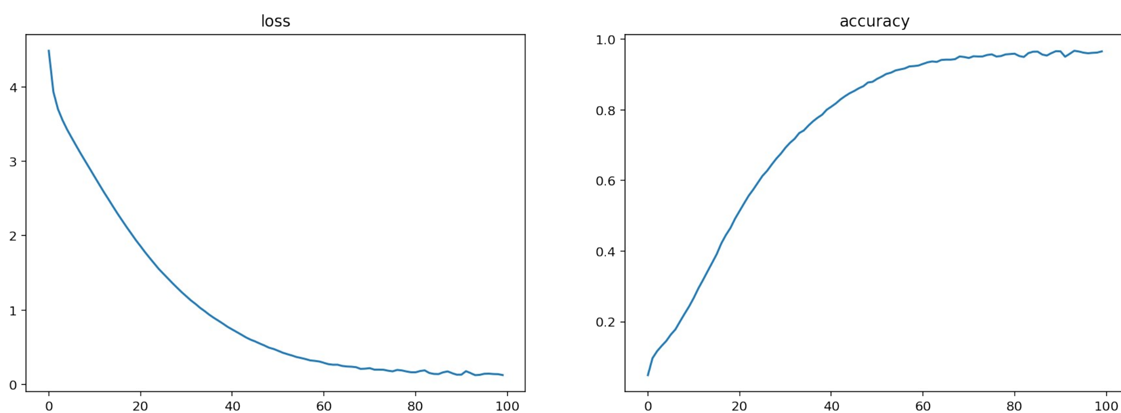

Figura 38: Gráficos de loss y accuracy

Los resultados obtenidos con esta representación se han generado de la misma forma que en los casos anteriores empleando un muestreo de temperatura y con una semilla inicial en este caso formada por los 10 primeros eventos *note on* de una pieza escogida aleatoriamente. Como se ha ignorado la duración de las notas, se ha fijado la duración de cada una de ellas a la duración de un pulso. Del mismo modo, se ha fijado la velocidad de todos los eventos *note on* de forma que suenen todos con la misma intensidad.

```
song_len = 500

selected_song = categorical_songs[np.random.randint(0, len(categorical_songs))]
start = selected_song[:ws]

for temp in [0.1, 0.2, 0.5, 0.8, 1.0, 1.2, 2]:

  song = np.zeros((song_len, n_vocab))
  song[:ws, :] = start

  pbar = ProgressBar(prefix='Generando muestra con temperatura {}...'.format(temp))

  for i in pbar(range(song_len - ws)):
    h = model.predict(song[i:i+ws].reshape(1, -1, n_vocab))
    s = sample(h, temp)
    song[i+ws, s] = 1

  num_song = np.argmax(song, axis=1)
  symbols_song = [int2sym[i] for i in num_song]
  sym2midi(symbols_song).save('/content/drive/My Drive/TFG/gen/gen_chopin_eq_{}.mid'.format(temp))
```

Figura 39: Código usado para generar música en formato MIDI ignorando la duración e intensidad de las notas

La música obtenida de esta forma no tiene ningún tipo de expresividad, pues todas las notas suenan con la misma intensidad y duran lo mismo además de no haber silencios de ningún tipo. Estos problemas son claras consecuencias de las limitaciones impuestas para poder generar música de forma sencilla, pero también parece que hay sobreajuste porque sin importar la temperatura todas las piezas generadas continúan de la misma forma o de una forma muy similar a partir de la semilla hasta llegar a un punto en el que divergen.

4.4 Generación empleando Performance RNN

Para esta representación, además de emplear la codificación descrita en el artículo original, se ha empleado la misma arquitectura. En este artículo, se propone una red de 3 capas LSTM con 512 células cada una. Al final de la red hay una capa densamente conectada con la función de activación softmax que dará la probabilidad de cada uno de los eventos.

En este artículo además emplean para el entrenamiento los archivos MIDI generados durante la interpretación de varias piezas por los concursantes de Piano e-Competiton,

un concurso de piano en el que las interpretaciones de los participantes quedan registradas tanto en audio como en MIDI. Este conjunto está formado por aproximadamente 1400 piezas que son además transportadas a distintas tonalidades para aumentar el conjunto de datos. Por razones de tiempo, se ha empleado para el entrenamiento el conjunto de piezas de Chopin empleado en el caso anterior, que transportado a distintas tonalidades se transforma en un conjunto de 720 piezas. En el caso original las piezas se dividen en secuencias de 30 segundos pero en este caso, cada una de las piezas se ha dividido en segmentos de 2000 pseudoeventos dando como resultado final un conjunto de 3540 secuencias. Cada una de las secuencias se recorre con una ventana de longitud 50, para tratar de predecir el siguiente evento. El entrenamiento se ha realizado durante 50 epochs.

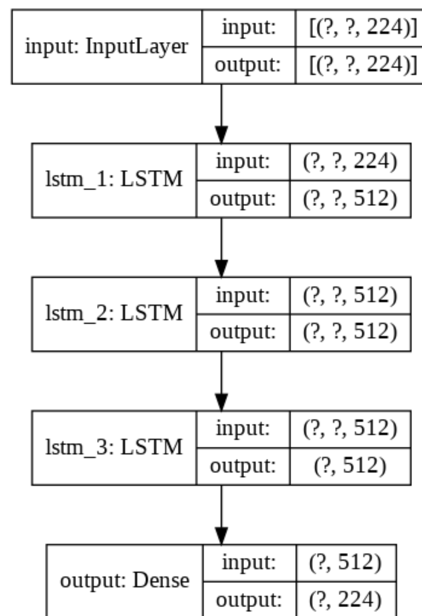

Figura 40: Arquitectura de Performance RNN

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_1 (LSTM)                (None, None, 512)         1509376
_________________________________________________________________
lstm_2 (LSTM)                (None, None, 512)         2099200
_________________________________________________________________
lstm_3 (LSTM)                (None, 512)               2099200
_________________________________________________________________
output (Dense)               (None, 224)               114912
=================================================================
Total params: 5,822,688
Trainable params: 5,822,688
Non-trainable params: 0
_________________________________________________________________
```

Figura 41: .summary() de la red

```
for temp in [0.1, 0.2, 0.5, 0.8, 1, 1.2, 2]:
  song_len = 2000
  ws = 10

  r = encoded[0]

  song = np.zeros((1, song_len, n_features), dtype=int)

  r = to_categorical(r[:ws], num_classes=n_features)
  song[:, :ws, :] = r

  pbar = ProgressBar(prefix='Generando muestra con temperatura {}...'.format(temp))
  for i in pbar(range(song_len - ws)):
    h = model.predict(song[:, :i+ws, :])[0]
    next_val = sample(h, temp)
    song[:, i+ws, next_val] = 1

  song_decoded = np.argmax(song[0], axis=1)

  tuples = []
  for n in song_decoded:
    tuples.append(num2sym[n])

  tuples = np.array(tuples)
  decoded = utils.decode_zigzag(tuples)
  decoded.save('/content/drive/My Drive/TFG/gen/performance_rnn_train_{}.mid'.format(temp))
```

Figura 42: Código para generar secuencias MIDI empleando la codificación de Performance RNN

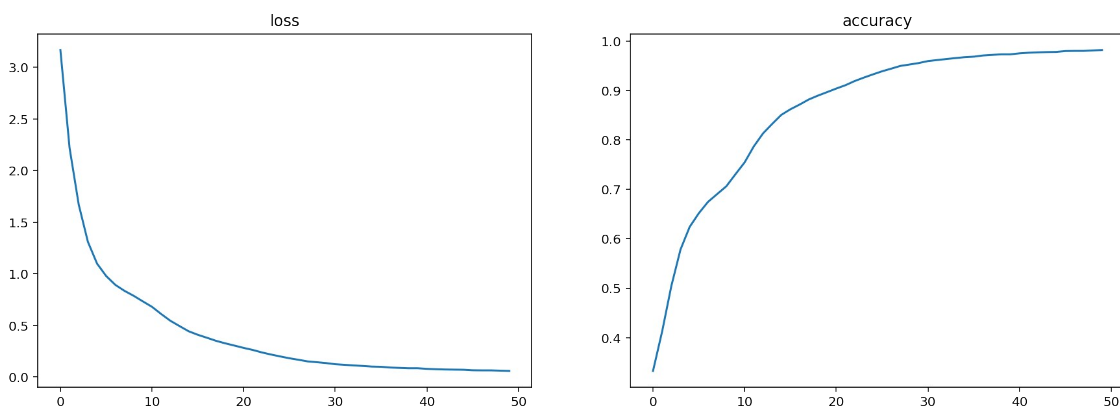

Figura 43: Loss y accuracy para Performace RNN

Esta representación permite generar música con mayor expresividad que en los casos anteriores porque se puede controlar la intensidad con la que suena cada nota y permite silencios a diferencia de la codificación anterior. Sin embargo, no siempre se generan eventos *note off* para cada evento *note on*, lo que conlleva a que algunas notas se mantengan sonando hasta el final de la pieza. Esto es un inconveniente doble, pues una nota que se mantiene sonando puede causar disonancias con el resto de notas y además no puede ser tocada de nuevo mientras suena. Para solucionarlo es necesario tener en cuenta que notas están sonando en cada momento y forzar un

evento *note off* transcurridos 5 segundos desde el note on si todavía no ha llegado su evento *note off* correspondiente.

4.5 Generación empleando audio

La última representación que se ha evaluado es la señal de audio crudo. Para ello se ha empleado una implementación de WaveNet que ha sido necesario adaptar a partir de una implementación en una versión antigua de Keras. Para ello, se han reemplazado las capas AtrousConvolution1D por capas Conv1D con su valor de dilation_rate correspondiente, las funciones add y merge por las capas Add y Multiply y además se ha sustituido el código empleado para asignar el valor de cada muestra a una clase por funciones que implementan las fórmulas 11 y 12 y se ha empleado una profundidad de bit con valor 8.

En el repositorio original[14] se indica que se ha entrenado con el dataset Librispeech. Este dataset contiene unas 1000 horas de texto hablado en inglés. Dada la limitación de tiempo, en este caso se ha evaluado empleando un conjunto mucho más reducido compuesto por una única canción. Esta canción se ha obtenido convirtiendo a wav uno de los archivos MIDI del dataset empleado en los puntos 4.3 y 4.4. El modelo se ha intentado entrenar por 2000 epochs, pero aun reduciendo el conjunto de datos de esta forma cada uno de los epochs tarda aproximadamente 15 minutos en una GPU Tesla V100.

Debido a su alto coste en tiempo y a la limitación de tiempo de Google Colab a 12 horas en su versión gratuita y 24 en la versión de pago no se podría pasar de 48-96 epochs durante el entrenamiento. En este caso, el entrenamiento se ha interrumpido al alcanzar el límite de tiempo tras 78 epochs. El entrenamiento comienza con un accuracy de 0.23 y finaliza con uno de 0.57, aparentemente habiendo llegado a converger en este punto. Se puede escuchar el progreso de cada epoch, puesto que al finalizar cada uno de ellos se ha generado una muestra de audio de 1 segundo de duración. Al escuchar las muestras, se puede apreciar como al principio todo lo que se puede oír es ruido, pero poco a poco comienza a escucharse una frecuencia un poco mas aguda de fondo, aún siendo bastante predominante el ruido. Al finalizar el entrenamiento, se ha generado una muestra de 1 minuto de duración para poder comprobar si es capaz de generar algo mas que un tono constante, ya que en las muestras generadas anteriormente la duración es tan corta que no hay tiempo para que haya variaciones. Para generar esta muestra de 1 minuto son necesarias 4 horas y 27 minutos. El audio generado es igual al del último epoch, compuesto por ruido y un tono constante de fondo ligeramente audible.

14 https://github.com/usernaamee/keras-wavenet

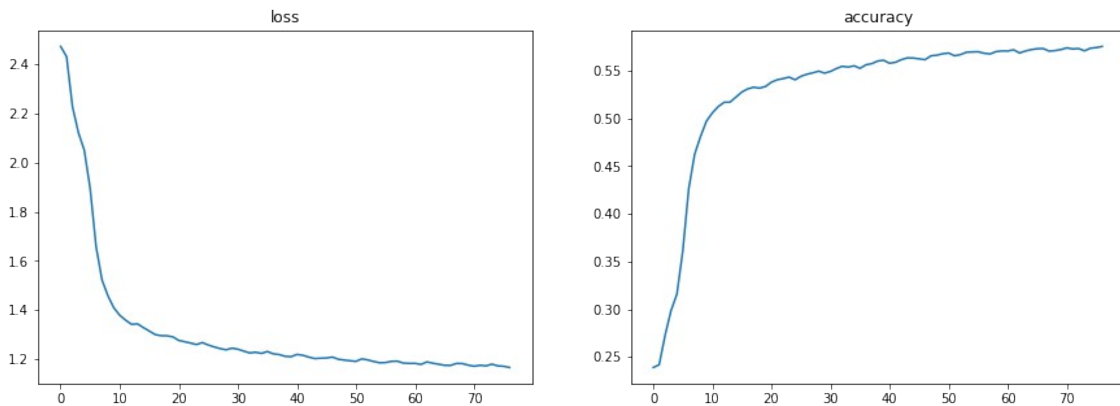

Figura 44: loss y accuracy en WaveNet

Con estos resultados, se puede asegurar que esta representación no es la más adecuada para esta tarea debido a la gran cantidad de recursos necesarios para entrenar un modelo que sea capaz de generar música de manera satisfactoria, además del tiempo necesario para entrenar el modelo y generar las muestras. El ruido es también un factor importante por el que descartar esta representación, puesto que no acaba de desaparecer ni de atenuarse en ningún momento, dificultando la escucha.

5 Otras representaciones

Hay otras representaciones que no se han evaluado, ya sea por la falta de tiempo o por la poca literatura al respecto sobre ellas. Es el caso de la transformada de Fourier o el espectrograma que apenas se mencionan en la generación de audio, pero sí que son adecuadas para tareas como la clasificación y transcripción.

5.1 Transformada de Fourier

La transformada de Fourier permite transformar una señal en el dominio del tiempo en señales en el dominio de la frecuencia. Este proceso es reversible aplicando la transformada inversa de Fourier. Existen multitud de tipos de transformadas de Fourier, siendo la necesaria en este caso la transformada discreta de Fourier, puesto que la señal de audio está formada por muestras discretas.

Esta transformación permite descomponer la señal original en varias señales, cada una de ellas con una frecuencia fija. De esta forma el análisis de la señal se simplifica al conocer cuales son las frecuencias que la forman y cuánto contribuye cada una de ellas a la formación de la señal original. Aún así, no supone una reducción en las dimensiones de los datos y cualquier modelo que se quiera entrenar con esta representación de audio tendrá en la práctica los mismos problemas que aparecen al emplear la señal sin ningún tipo de transformación.

5.2 Espectrograma

El espectrograma no es más que una forma de visualizar la transformada discreta de Fourier. Para construirlo, se recorre la señal con una ventana de un tamaño fijo y se calcula la transformada discreta de los valores dentro de la ventana. A continuación, se desplaza la ventana de forma que la nueva ventana quede solapada con la ventana anterior. Como resultado se obtiene una matriz de números complejos que se puede descomponer en dos imágenes: el espectrograma de magnitud y el espectrograma de fase.

El espectrograma de magnitud es lo que todos conocemos como espectrograma y es el que se suele visualizar puesto que es el que más información contiene. El espectrograma de fase es en cambio una imagen que a simple vista no se puede distinguir del ruido y suele ser omitido. Es posible recuperar de forma aproximada la señal original a partir del espectrograma de magnitud, pero sin el espectrograma de fase el resultado suena notablemente distinto. El valor de la fase se puede aproximar mediante algoritmos de reconstrucción de fase, pero aún así la información por cada unidad de tiempo en un espectrograma es mucho mayor que en una señal de audio. Es por ello que esta representación tampoco es de las más adecuadas para la generación pero sí que da buenos resultados para otras tareas como la clasificación o la transcripción de música.

Para el problema de la transcripción en concreto es necesario además afrontar el problema de la detección de *onsets*, un problema que no está presente en la generación de música. Este problema consiste en detectar en qué momento comienza a sonar una nota y también se puede resolver con *deep learning* aunque la forma más sencilla es mediante técnicas algorítmicas. Una de las arquitecturas que se pueden usar para transcribir música y emplean *deep learning* tanto para la detección de *onsets* como para la transcripción es la propuesta en [7].

6 Conclusiones y líneas futuras

En este trabajo se han evaluado diferentes alternativas para la codificación de música y se han generado arquitecturas de red para la generación automática de música, utilizando principalmente las codificaciones ABC y MIDI. En cuanto a los resultados obtenidos en la generación de música, se puede observar que en algunos casos parece haber tanto sobreajuste que llega a copiar secuencias del conjunto de entrenamiento. En los problemas de generación de secuencias, es deseable que haya algo de sobreajuste porque el objetivo es que dado un conjunto de secuencias, las secuencias generadas sean similares a las del conjunto original, pero en ningún caso se espera que las secuencias sean completamente copiadas. Este es uno de los puntos a mejorar en el futuro puesto que en ningún caso se ha comprobado que los resultados sean

originales y no hayan copiado nada. Es por ello que sería conveniente emplear algún tipo de métrica o metodología que permita evaluar si se han copiado partes del conjunto de entrenamiento.

Otro de los puntos a mejorar es la poca coherencia en los resultados en la generación carácter a carácter empleando notación ABC. Es posible que los resultados no sean del todo buenos porque se han incluido cabeceras innecesarias, que no aportan nada a la música como tal. Además se puede repetir el experimento transponiendo todas las piezas a la misma tonalidad para tener un punto de referencia común y la música generada no contenga ninguna nota fuera de la escala ni aparezcan sonidos extraños causados por notas disonantes. Esto no debería ser muy difícil puesto que todas las piezas en esta notación incluyen la tonalidad en la que se encuentran, y con esta información es sencillo transportarlas a una tonalidad común, como puede ser la de do mayor/la menor que no tiene ninguna alteración y permitiría que la música resultante tuviera un sonido agradable aún cuando no se haya especificado un campo K en las cabeceras.

La generación con archivos MIDI en general produce resultados satisfactorios, pero tiene también muchas otras representaciones que no se han evaluado. En este trabajo se ha mencionado la representación *piano roll* mayormente porque es una manera cómoda de visualizar los ficheros MIDI y es la que se ha empleado para la explicación del resto de codificaciones, pero es posible también utilizarla para generar música. Hay multitud de representaciones más, pero una que parece tener buenos resultados y no se ha evaluado es la empleada en [8], que funciona de la misma forma que el rollo de piano pero redice la dispersión de los datos empleando un número pequeño de filas a los que llama *streams* en lugar de una fila por cada posible nota, y asigna las notas que suenan simultáneamente a una misma posición de la misma forma que el *piano roll*, situando en el *stream* más bajo las notas más graves y en los más altos las más altas. De esta forma quedan todas las notas que suenan a la vez localizadas en una misma columna formada por los instantes de los distintos streams y se localizan los sonidos en función de su tonalidad, favoreciendo que se aprendan las relaciones entre tonos cercanos entre sí.

Se ha mencionado también que Performance RNN se podría extender para trabajar con múltiples instrumentos a la vez como lo hace BandNet, pero no se ha podido hacer por falta de tiempo.

MIDI es posiblemente el formato más conocido y empleado para la representación simbólica de música en *deep learning* y en consecuencia muchas de las arquitecturas existentes para representaciones simbólicas trabajan con MIDI. No obstante en este trabajo solamente se han evaluado los resultados de las distintas representaciones de un archivo MIDI en una de ellas, Performance RNN, pero se pueden evaluar otras

muchas como DeepBach o MusicVAE. En [9] se propone una red con su respectiva representación que aparentemente da muy buenos resultados.

Finalmente, para la generación de música empleando la señal de audio se pueden probar otras arquitecturas. Existen otras muchas que trabajan con la señal de audio como GANSynth, MelNet o la mencionada anteriormente SampleRNN. Además se puede considerar estudiar otras representaciones alternativas como la transformada de Fourier para generación de audio. Aparentemente este campo todavía no está muy estudiado y la mayoría de arquitecturas que trabajan con señales de audio lo hacen con el audio crudo y no emplean este tipo de transformaciones, pues es necesario recuperar la magnitud y la fase y mientras la magnitud se puede generar de forma relativamente simple, el problema de la recuperación de fase es todavía un problema abierto y requiere ser resuelto para poder reconstruir la representación en función del tiempo. Recientemente han aparecido arquitecturas como TiFGAN [10] capaces de trabajar con este tipo de representaciones.

7 Bibliografía

[1] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, Koray Kavukcuoglu *WaveNet: A Generative Model for Raw Audio* (2016)

[2] Soroush Mehri, Kundan Kumar, Ishaan Gulrajani, Rithesh Kumar, Shubham Jain, Jose Sotelo, Aaron Courville, Yoshua Bengio *SampleRNN: An Unconditional End-to-End Neural Audio Generation Model* (2016)

[3] Oore, S., Simon, I., Dieleman, S. *et al. This time with feeling: learning expressive musical performance*. Neural Comput & Applic 32, 955–967 (2020)

[4] Yichao Zhou, Wei Chu, Sam Young, Xin Chen *BandNet: A Neural Network-Based Multi-Instrument Beatles-Style MIDI Music Composition Machine (2018)*

[5] Sepp Hochreiter, Jürgen Schmidhuber *Long Short-term Memory*. Neural computation (1997)

[6] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, Yoshua Bengio *Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling* (2014)

[7] Curtis Hawthorne, Erich Elsen, Jialin Song, Adam Roberts, Ian Simon, Colin Raffel, Jesse Engel, Sageev Oore, Douglas Eck *Onsets and Frames: Dual-Objective Piano Transcription* (2018)

[8] Harish Kumar, Balaraman Ravindran *Polyphonic Music Composition with LSTM Neural Networks and Reinforcement Learning* (2019)

[9] Daniel D. Johnson *Composing Music With Recurrent Neural Networks* (2015)

[10] Andrés Marafioti, Nicki Holighaus, Nathanaël Perraudin, Piotr Majdak *Adversarial Generation of Time-Frequency Features* (2019)