Universidad Zaragoza
1542

Trabajo Fin de Grado

Composición Musical mediante una Red Neuronal

Musical Composition with a Neural Network

Autor

Carlos Hernández Oliván

Director

José Ramón Beltrán Blázquez

ESCUELA DE INGENIERÍA Y ARQUITECTURA

2017

TRABAJOS DE FIN DE GRADO / FIN DE MÁSTER

Escuela de Ingeniería y Arquitectura Universidad Zaragoza

DECLARACIÓN DE AUTORÍA Y ORIGINALIDAD

(Este documento debe acompañar al Trabajo Fin de Grado (TFG)/Trabajo Fin de Máster (TFM) cuando sea depositado para su evaluación)

D./Dª. CARLOS HERNÁNDEZ OLIVÁN,

con nº de DNI 25201835 E en aplicación de lo dispuesto en el art. 14 (Derechos de autor) del Acuerdo de 11 de septiembre de 2014, del Consejo de Gobierno, por el que se aprueba el Reglamento de los TFG y TFM de la Universidad de Zaragoza,

Declaro que el presente Trabajo de Fin de (Grado/Máster) Grado de Ingeniería de Tecnologías Industriales, (Título del Trabajo) Composición Musical Mediante una Red Neuronal

es de mi autoría y es original, no habiéndose utilizado fuente sin ser citada debidamente.

Zaragoza, 19 Junio 2017

Fdo: CARLOS HERNÁNDEZ OLIVÁN

A mi familia que tanto me ha apoyado durante estos años, a mi profesor de viola del Conservatorio Profesional de Música de Zaragoza, D. Juan Luis Arcos y al director del mismo, D. Darío Sierra, que me inculcaron el amor por la música, y al director de este trabajo, D. José Ramón Beltrán por su apoyo y dedicación.

Y en especial a mi abuelo, que despertó en mí el amor por la ingeniería.

RESUMEN

En este trabajo se ha realizado el análisis y la comprobación de una red neuronal existente previamente, que compone música mediante técnicas de Aprendizaje Profundo (*Deep Learning*). El objetivo del trabajo ha sido realizar un proceso de composición musical automática de una *Sonata*. Para ello, tras la comprensión de la red, la iniciación en el lenguaje de programación Python 2 y la generación de una base de datos en formato MIDI mediante los programas Finale y Logic Pro X, se ha diseñado una red optima para la composición, entrenándola y evaluando sus resultados.

Además, se exponen tres anexos con el fin de presentar las distintas formas de generar música algorítmica, de definir los conceptos *Deep Learning* y *Big Data*, así como de situar en un contexto histórico la Forma Sonata y repasar conceptos fundamentales del lenguaje musical necesarios para comprender tanto la entrada, como la salida de la red. En el cuarto y último anexo se muestra la partitura de la obra resultante del entrenamiento de la red.

ABSTRACT

In this project it has been done an analysis and checking of a neural network already designed, which composes music through Deep Learning techniques. For that, after the understanding of the network, the initialization in the Python 2 language and the generation of a data base in MIDI format through Finale and Logic Pro X programs, an optimum network has being designed for composition, training it and evaluating its results.

Besides, four annexes are exposed in order to present the different ways of generating algorithmic music, define Deep Learning and Big Data concepts, as well as situate in an historic context the Sonata Form and review fundamental concepts of musical language, which are necessary to understand the input and the output of the network, and show the score of the output piece after the network training.

ÍNDICE

ANEXOS

ÍNDICE DE FIGURAS

ÍNDICE DE EXPRESIONES

ÍNDICE DE TABLAS

Capítulo 1

Introducción

En este capítulo se realiza una introducción al trabajo exponiendo los objetivos fijados al comienzo del mismo, así como la presentación de las redes neuronales y su importancia en el presente y futuro de la ingeniería.

1.1. Motivación

El estudio del ser humano y sus capacidades ha sido desde siempre uno de los principales objetos de la investigación. Desde el primer computador diseñado por Alan Turing para descifrar códigos hasta los avances actuales en *Deep Learning*, el objetivo principal siempre ha sido el de construir máquinas capaces de realizar las mismas tareas que un ser humano, de una manera mucho más rápida y eficaz. El término de red neuronal nace de la evolución de estas máquinas y algoritmos, para acercarse cada vez más al pensamiento y razonamiento humano.

El cerebro humano contiene más de cien mil millones de neuronas y 10^{14} sinapsis en el sistema nervioso. El objetivo principal de las redes neuronales es desarrollar operaciones de síntesis y procesamiento de información. Las redes neuronales profundas siguen un aprendizaje profundo también conocido como *Deep Learning*. Actualmente el campo de investigación del *Deep Learning* es amplísimo, dado que existen infinidad de aplicaciones posibles a desarrollar mediante estos algoritmos, y una de ellas es la estudiada en este trabajo.

Existen dos formas de generar música de manera automática, mediante música algorítmica y mediante redes neuronales. Este trabajo se centra en la forma de componer música con algoritmos profundos, que está en un estado muy incipiente de desarrollo. Como estudiante de música durante muchos años y de ingeniería estos últimos, el reto de poder componer música mediante estas herramientas como sustitución a lo que haría un ser humano, además de ser una línea de investigación en nacimiento, es un motivación extra para la realización de este trabajo.

1.2. Objeto y Alcance

El objetivo de este trabajo es el comienzo de una nueva línea de investigaciónen composición automática que todavía no está consolidada, dado que dichos procesos de composición automática se vienen realizando desde hace tiempo mediante algoritmos clásicos y no con estrategias de aprendizaje profundo *Deep Learning*.

Concretamente, el objetivo de este trabajo es componer mediante una red neuronal capaz de generar notas musicales, la forma más compleja que existe dentro de la música clásica: la Forma Sonata.

Dado que resulta muy complejo generar dicha estructura mediante una red que obtiene notas musicales sin un sentido musical a priori, y sin una base de datos bien organizada desde el punto de la historia de la música y la teoría musical, se han fijado los siguientes objetivos para llevar a cabo dicha tarea:

- Estudio general de redes neuronales y la comprensión de una red ya diseñada.
- Obtención de una base de datos MIDI de las obras a entrenar.

- Comprensión e inicio en la programación del lenguaje Python y uso de librerías como Theano, Numpy, etc.
- Entrenamiento de la red para la determinación del número de neuronas y epochs óptimos tras varios entrenamientos.
- Realización de la interfaz gráfica de usuario para hacer más accesible el Software a personas sin conocimientos de programación.
- Análisis y valoración de los resultados obtenidos.

1.3. Descripción de la Memoria

La memoria de este trabajo está compuesta por seis capítulos a los que complementan cuatro anexos.

En el segundo capítulo se expone el concepto de red neuronal, describiendo la estructura y método de entrenamiento del tipo de red analizada en el trabajo, las redes Long Short-Term Memory.

En el tercer capítulo se describe la red neuronal tomada como referencia y se exponen los cambios realizados en dicha red para su entrenamiento posterior en función del número de neuronas y *epochs*. Además, se expone la interfaz gráfica para el usuario.

En el cuarto capítulo se recopilan y analizan los resultados obtenidos del entrenamiento de la red neuronal. Además, se valoran los resultados mediante un criterio propuesto en el propio capítulo.

En el quinto capítulo se exponen las concusiones obtenidas al finalizar el trabajo y las líneas futuras que quedan pendientes en la investigación de la composición musical con redes neuronales.

En el Anexo I se describen los tipos de composición algorítmica que han ido surgiendo a lo largo de los años, puesto que es la base que sigue la línea de investigación de la composición musical de manera automática.

En el Anexo II, puesto que son necesarios unos conceptos mínimos de lenguaje musical e historia de la música, se hace una breve introducción a estas dos ramas de la música para que se sepa valorar de una forma más o menos objetiva los resultados del entrenamiento de la red neuronal.

En el Anexo III se definen los conceptos de *Deep Learning* y *Big Data*, imprescindibles para comprender el proceso de aprendizaje profundo y las estructuras de datos utilizadas para el análisis de grandes cantidades de información.

En el Anexo IV se recoge la partitura de la obra resultante del entrenamiento de la red.

1.4. Contexto Histórico

La Inteligencia Artificial (AI) es un área de la ciencia que estudia aquellas capacidades que se consideran "inteligentes". La inteligencia artificial involucra una gran cantidad de áreas genéricas, como el aprendizaje, la percepción o la resolución de problemas, y específicas, como el diagnóstico de enfermedades, conducción de vehículos, etc. En definitiva, la inteligencia artificial es un campo de estudio que busca emular la capacidad del pensamiento humano en modelos computacionales.

Para la existencia de la IA es necesario un Hardware y herramientas desarrolladoras de programas de IA (Software). Dichas herramientas de Software nacieron en los años 1940 de la mano de Warren McCulloch y Walter Pitts, quienes realizaron los primeros modelos de redes neuronales.

Las **redes neuronales** fueron en su origen una simulación abstracta de los sistemas nerviosos biológicos, constituidos por un conjunto de unidades de procesamiento llamadas neuronas o nodos conectados entre ellos. Las neuronas reciben una serie de entradas, que se propagan por las conexiones entre las neuronas a través de la red, y emiten una salida. En la figura 1 se muestra un esquema general del proceso llevado a cabo en el diseño de una red neuronal artificial.

Estas redes neuronales pueden aprender mediante algoritmos de aprendizaje profundo o ***Deep Learning***, que permiten predecir sucesos, identificar imágenes, sugerir rutas, etc, basándose en datos introducidos como entrada a la red, lo que hace a estos softwares lo más parecido a una mente humana.

Para el funcionamiento de este tipo de Software se necesitan grandes cantidades de información y datos, utilizados como entrada para el cálculo o predicción de sucesos. Este almacenamiento de datos masivos se conoce como ***Big Data***.

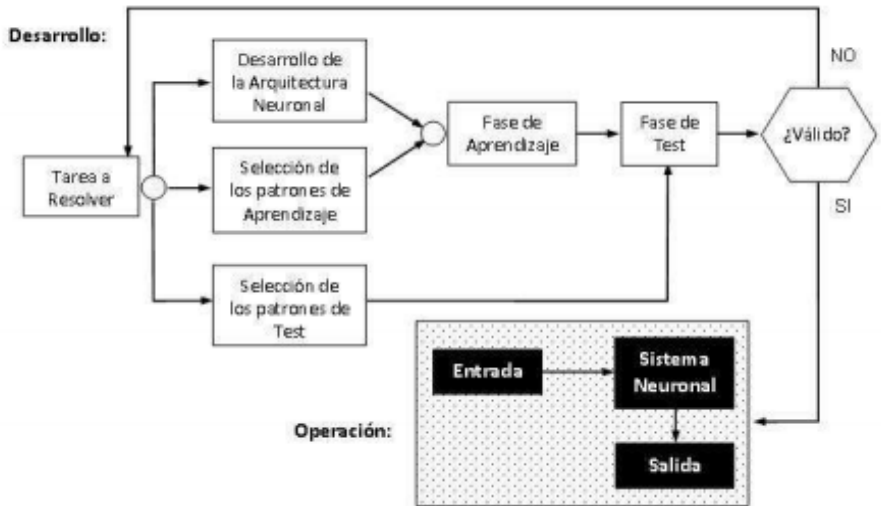


Figura 1. Esquema de una RNA [1]

Capítulo 2

Redes Neuronales Artificiales

En este capítulo se introduce el concepto de red neuronal artificial, exponiendo los diferentes tipos de redes neuronales que existen atendiendo a la bibliografía, así como los tipos de entrenamiento más utilizados. Se profundizará en las redes Long Short-Term Memory, tipo de red utilizada en el trabajo para generar música.

2.1. Introducción

El cerebro humano posee entre 50 y 100 mil millones de neuronas. Las células transmiten señales nerviosas a través de hasta 1.000 billones de conexiones sinápticas. Las redes neuronales artificiales tratan de imitar el sistema nervioso, estableciendo conexiones que transmiten la información entre capas de neuronas artificiales.

Los primeros modelos de redes neuronales datan del año 1943, de la mano de los neurólogos Warren McCulloch y Walter Pitts. En 1949, Donald Hebb desarrolló la "regla de Hebb" donde expone sus ideas sobre el aprendizaje neuronal, y en 1958, Rosenblatt desarrolló el perceptrón simple. La primera aplicación industrial de redes neuronales fue ADALINE, desarrollada en 1960 por Widrow y Hoff.

Las redes neuronales [2] destacan por su gran capacidad de generalización, aunque poseen otras propiedades destacables como la no linealidad, ya que pueden ser lineales o no lineales, la adaptabilidad que es la capacidad de reajustarse ante cambios en el entorno, y la tolerancia ante fallos.

Las aplicaciones de las redes neuronales son amplísimas. Algunas de las ellas son la predicción de enfermedades, reconocimiento de imágenes y sonidos, y composición musical, entre otras.

2.2. Concepto de Red Neuronal Artificial

Las redes neuronales son un paradigma de aprendizaje y procesamiento automático basado en el sistema nervioso. Dichas redes forman un sistema de interconexión de neuronas, que reciben como entrada señales continuas o discretas y transmiten la información a las neuronas conectadas a ellas para producir una salida.

2.2.1. Representación Gráfica

Para representar una red neuronal [2] se han de ilustrar los estados en un tiempo *t* representados por nodos. La representación de un sistema dinámico se muestra en la figura X, donde s es el estado del sistema en un tiempo *t*. En el apartado 2.4.1. se muestra la representación de las redes neuronales recurrentes, tipo de red utilizada en este trabajo.

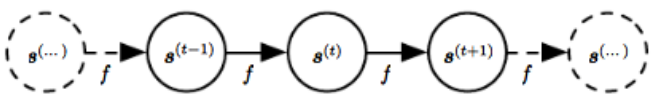


Figura 2. Representación de un sistema dinámico [3]

2.2.2. Perceptrón y Redes Multicapa

Un **perceptrón** [3] es un tipo de red neuronal que toma varias entradas binarias $x_i[t]$ que suministran a la neurona datos del entorno, y produce una sola salida, también binaria. Cada entrada posee su sinapsis, caracterizada cada una por un **peso** W_{ji}. Cada entrada tiene su propio peso relativo, que contiene la mayor parte del conocimiento que la red tiene sobre la tarea en cuestión. El valor de estos pesos se modifica mediante el entrenamiento de la red. La salida está determinada por la suma ponderada de $\Sigma_j W_{ji} x_j$, conocida como **función de propagación**, y la adición de un término constante denominado ***bias*** o **sesgo**, W_{ji}, que limita la amplitud de la salida de la neurona.

Para comprimir el resultado de la suma $\Sigma_j W_{ji} x_j$ entre 0 y 1, se utiliza una **función de transferencia** o **función de activación, *f***. Una de las funciones de activación más utilizadas en las redes neuronales es la función sigmoide, no lineal, cuya representación y expresión aparecen en la figura 2.

$$f(z) = \frac{1}{1 + exp(-z)}$$

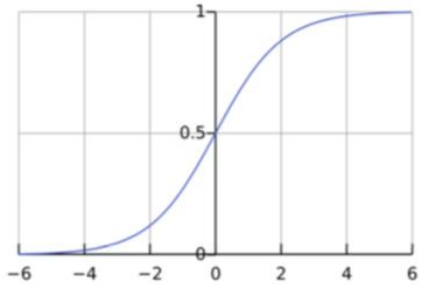


Figura 3. Función Sigmoide

Otras funciones, como la tangente hiperbólica permiten comprimir la suma $\Sigma_j w_j x_j$ entre -1 y 1.

La operación de una neurona [2] puede describirse mediante la expresión (1), que determina su activación en el instante t+1:

$$x_j[t+1] = f\left(\sum_{i=1}^{n} W_{ji} x_i[t] + W_j\right) \quad (1)$$

Como se ha descrito anteriormente, el perceptrón simple recibe una o más entradas y genera una única salida. En la figura 3 se muestra un ejemplo de perceptrón simple con 5 entradas.

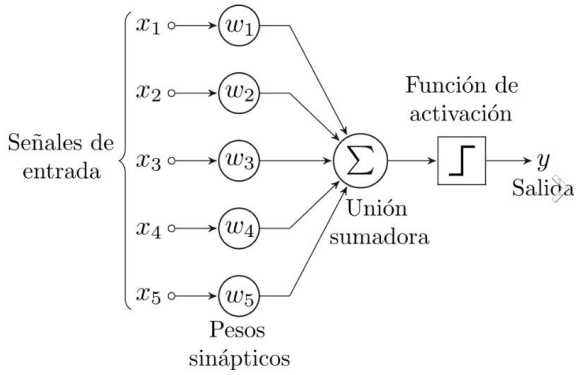


Figura 4. Diagrama de un perceptrón simple con 5 entradas [4]

En este trabajo, se va a utilizar una estructura más compleja que la del perceptrón simple. Dicha estructura va a estar formada por un conjunto de neuronas o nodos, que formarán una capa (figura 4). Cada entrada irá conectada a cada una de las neuronas de la capa siguiente.

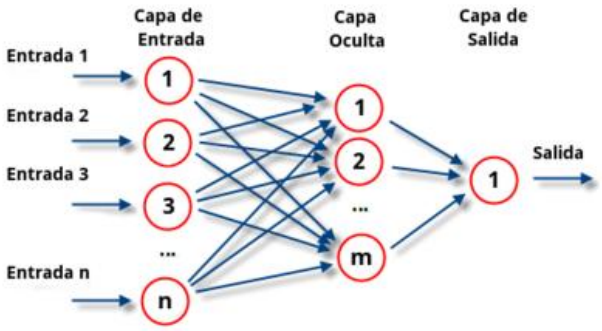


Figura 5. RNA con una capa oculta de neuronas [5]

Para completar la estructura total de la red, se conectarán varias capas de neuronas, las cuales pertenecerán a estados intermedios de cálculo y que, por tanto, no pertenecerán a la salida de la red. La red neuronal quedará formada por la capa de entrada o *input layer*, la capa de salida o *output layer* y las capas intermedias u ocultas *hidden layers*.

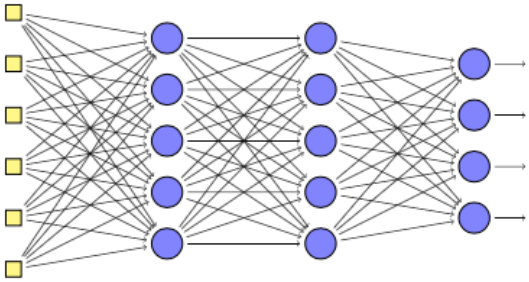


Figura 6. RNA Multicapa [6]

2.3. Entrenamiento de Redes Neuronales

El entrenamiento de una red neuronal multicapa [7] se realiza mediante un proceso de aprendizaje. Para la realización de este proceso se debe definir el número de neuronas de la capa de entrada, el número de capas ocultas y neuronas de cada una de ellas, neuronas en la capa de salida y funciones de transferencia en cada capa.

No existe un método para determinar el número de neuronas y capas ocultas, aunque existen cotas máximas y mínimas. Normalmente, cuantas más capas mejor es el aprendizaje, pero más lento es el procesamiento de un *epoch*. Un *epoch* es un ciclo de entrenamiento, es decir, un solo recorrido en la dirección de computación de la red y otro recorrido en sentido contrario de todos los patrones de entrenamiento.

Cada patrón de entrenamiento se propaga a través de la red, y la salida se compara con el objetivo marcado. Se calcula el error, diferencia de la salida con el valor deseado, y se recalculan los pesos en cada *epoch* mediante el algoritmo

backpropagation, que es un método de aprendizaje supervisado de **gradiente descendente**, que toma el error a la salida y se calcula en la dirección a la entrada, es decir, en dirección opuesta a la dirección de computación, de ahí el nombre *backpropagation* o propagación hacia atrás. Durante el entrenamiento se logrará así disminuir el error cuadrático medio.

$$E_p = \frac{1}{2}\Sigma(d_{pj} - y_{pj})^2 \tag{2}$$

El ajuste de los pesos se realiza con las expresiones (3) y (4) conocidas como la regla delta:

$$\Delta_P w_{ij} = \eta\big(d_{pj} - y_{pj}\big)x_{pi} = \eta\delta_{pj}x_{pi} \tag{3}$$

$$\delta_{pj} = (d_{pj} - y_{pj}) \tag{4}$$

Esta regla delta se realiza de acuerdo con el gradiente descendente, aunque esto es sólo cierto para variaciones nulas o muy pequeñas en los pesos durante las iteraciones. Cuando existen capas ocultas este proceso no se realiza de la misma forma, y se toma de partida una función de activación no creciente y diferenciable. El proceso se resume en las expresiones (5) y (6), que junto con la función de activación sigmoide se utilizan en el algoritmo *backpropagation* o propagación hacia atrás.

$$\Delta_P w_{ji} = \eta\delta_{pj}x_{pi} \quad con \quad \delta_{pj} = \big(d_{pj} - y_{pj}\big)\, f'_j(z_{pj}) \tag{5}$$

$$\delta_{pj} = f'_j\left(z_{pj}\right)\Sigma\,\delta_{pk}w_{kj} \tag{6}$$

Dado que en las redes multicapa la superficie del error no es cuadrática, la velocidad de convergencia puede incrementarse por la variación de la tasa de aprendizaje en cada parte de la superficie del error, y para modificar esta tasa de aprendizaje se siguen unas reglas heurísticas.

2.4. Redes Recurrentes

Las redes recurrentes son un tipo de red neuronal que se diferencian por formar ciclos o bucles, denominados conexiones recurrentes. Las conexiones recurrentes pueden ser de una neurona con ella misma (retroalimentación), entre neuronas de una misma capa, o entre neuronas de una capa con neuronas de la capa anterior.

Estas conexiones hacen que la complejidad de la red sea mayor, ya que aumenta el número de pesos de la red, y las activaciones no sólo dependen de las activaciones de la capa anterior, sino también de la activación de cualquier neurona conectada a la propia neurona o incluso de ella misma.

2.4.1. Representación Gráfica

Existen dos formas de representar las redes neuronales recurrentes. En la figura 7 se muestran las dos formas, una representando la recurrencia en forma de realimentación y la otra mediante conexiones en la dirección de computación, donde el parámetro *x* es la entrada de la red y *h* el estado en un tiempo *t*.

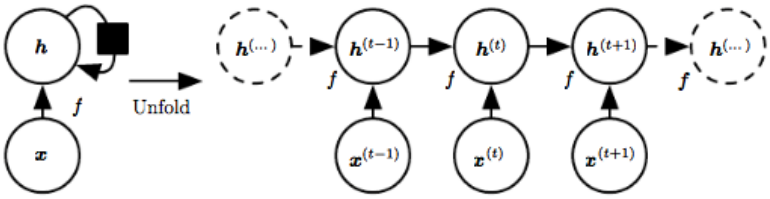


Figura 7. Representación de una red neuronal recurrente [8]

Desglosando el gráfico de la figura anterior se obtiene una representación donde el parámetro *y* es el objetivo, *U*, *W* y *V* los pesos entre la capa de entrada y la primera capa oculta, entre la primera capa oculta y la segunda, y entre la segunda capa oculta y la salida respectivamente. *L* es la pérdida o error que mide la distancia entre la salida *o* y objetivo *y*.

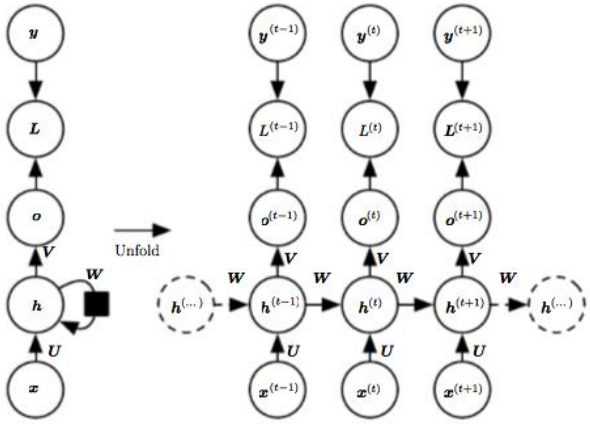


Figura 8. Representación de una red neuronal recurrente [8]

2.4.2. Arquitectura

Existen muchos tipos de redes recurrentes, entre los que se encuentran: redes totalmente recurrentes *fully recurrent networks*, redes recurrentes recursivas, redes de Hopfield, redes de Elman, redes de Jordan, *echo state networks*, *long short-term memory* (LSTM), redes bidireccionales, redes perceptrón multicapa, etc. En este trabajo en concreto se va a utilizar una red LSTM biaxial.

2.4.3. Entrenamiento

Los algoritmos de entrenamiento se encargan de encontrar la configuración de los pesos de la red que resuelvan el problema con éxito. El entrenamiento puede ser supervisado o no supervisado. En el caso de la red neuronal estudiada en este trabajo, para minimizar al máximo el error total se utiliza el método del gradiente descendente bajo un aprendizaje no supervisado, ya que no se provee de una salida objetivo a la red.

Este método utiliza el algoritmo *backpropagation*, mencionado anteriormente donde las salidas se propagan hacia atrás partiendo de la capa de salida, hacia las neuronas de la capa oculta anterior. Capa a capa se repite el proceso hasta que todas las neuronas de la red hayan recibido una señal de error que describa su contribución relativa al error total. Posteriormente se actualizan los pesos basándose en esta señal de error para hacer que la red converja.

2.5. Redes Long Short-Term Memory (LSTM)

Las redes LSTM fueron propuestas en 1997 por Sepp Hochreiter y Jürgen Schmidhuber. Estas redes se utilizan para clasificar, procesar y predecir eventos.

El elemento básico de las redes LSTM es la célula de memoria o *memory-cell* (figura 9), que se compone de 4 elementos principales: una puerta de entrada, una neurona con retroalimentación, una puerta de salida y una puerta denominada *forget gate*. El valor del peso de la retroalimentación es 1, y esto asegura que el estado de la célula de memoria permanezca constante en cada paso de tiempo o *timestep*.

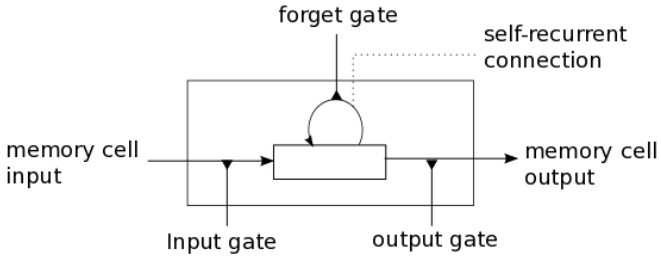


Figura 9. Célula de memoria [3]

Las puertas [2] son unidades multiplicativas con activación continua y son compartidas por todas las celdas que pertenecen a un mismo bloque de memoria. La **puerta de entrada** se encarga de permitir o impedir el acceso de los valores provenientes de la entrada y salidas de la red en el instante anterior al interior de la celda. La **puerta de salida** tiene una acción similar. La puerta ***forget gate*** puede modular la retroalimentación de la célula permitiéndole recordar u olvidar su estado previo, cuando sea necesario.

El método de entrenamiento se realiza mediante el algoritmo *backpropagation* descrito en el apartado 2.3 de este capítulo. En este tipo de redes *feed-forward*, se necesita este algoritmo ya que es necesario adaptar los pesos no sólo de la capa de salida, sino de toda la red multicapa. En el caso de este trabajo, este algoritmo en lugar de utilizar una salida deseada para entrenar las neuronas ocultas como se ha comentado antes, utiliza la derivada del error que se va calculando capa a capa en dirección opuesta a la dirección de computación, así no se recurre al proceso de prueba y error en el que se basaba la regla delta, que comparaba la salida a un valor objetivo y a partir de ahí actualizaba los pesos. Este algoritmo, en definitiva, es el más eficiente para este tipo de redes y el que permite a la red generalizar para la obtención de unos mejores resultados.

Capítulo 3

Red Neuronal para la Composición Musical en Python.

En este capítulo se explica la estructura de una red neuronal recurrente en Python utilizando librerías como Theano, Numpy o Scipy. Se expone el desarrollo de una interfaz gráfica por medio de Tkinter que hace la red más sencilla de utilizar para el usuario.

Para la comprensión de la elección de la base de datos MIDI con la que se ha entrenado la red, se recomienda ver el Anexo 2, donde se realiza una breve introducción a la historia de la música y al análisis musical.

3.1. Base de Datos

El primer paso para el desarrollo de la red es la obtención de una base de datos. El tipo de archivos a emplear como entrada de la red serán archivos MIDI, ya que este tipo de archivos de secuenciación permite codificar lo que sucede en una partitura para introducir esa información en este caso en la red neuronal. En el siguiente apartado de este capítulo se introduce el concepto de Formato MIDI.

Dado que el objetivo del entrenamiento de la red es obtener música en forma sonata, se han obtenido [9, 10] los primeros movimientos de varias sonatas. Con el programa *Finale*, tras analizar todas las obras, se han dividido todos los archivos de cada una de ellas en: Tema A, Tema B, Desarrollo y Coda (para comprender por qué se han escogido estas partes se recomienda ver anexo II), y mediante el programa *Logic Pro X* con el editor de lista, se han editado de tal forma que las notas correspondientes a la melodía estén en el canal uno y el acompañamiento en el canal dos. Para generar la base de datos final, se exportan todas las partes de cada sonata como archivos MIDI y se sitúan en una carpeta de donde la red tomará dichos archivos como entrada.

Las sonatas elegidas son para piano, ya que es el instrumento capaz de interpretar una melodía y un acompañamiento al mismo tiempo para el que más obras se han compuesto. Como se detallará en el siguiente capítulo, las obras deberán estar compuestas sobre un compás de 4/4 para reducir el error que puede haber al utilizar distintos compases con distintas subdivisiones de tiempo.

La base de datos final comprende: 21 sonatas de Beethoven, 13 sonatas de Mozart, 13 sonatas de Schubert, todas ellas divididas en las cuatro partes mencionadas anteriormente y agrupadas por directorios según compositor y parte de la obra, es decir, hay un directorio "Mozart Temas A" donde se encuentran los Temas A de todas las obras de este autor, otro "Temas A" donde se encuentran todos los Temas A de los tres compositores, etc, formando un total de 16 directorios y un total de 188 archivos MIDI.

3.2. Formato MIDI

Para poder codificar los archivos de audio se ha utilizado el formato MIDI (Musical Instrument Digital Interface). El formato MIDI es un protocolo estándar de comunicaciones que permite representar la información musical en un fichero hexadecimal.

En el archivo MIDI (.mid) se define una base de tiempos que describe las acciones que el intérprete efectúa sobre el controlador para almacenar la información temporal: un número de tiempos por compás y la duración de cada tiempo de figura de la notación musical en unidades, por ejemplo, 1 para redondas, 4 para negras, etc (ver Anexo II).

3.2.1. Eventos MIDI

Un evento MIDI es una orden que recibe el controlador que la ejecuta. Se graban en pistas asociadas a distintos canales MIDI pudiendo ser secuenciados a través de un editor MIDI.

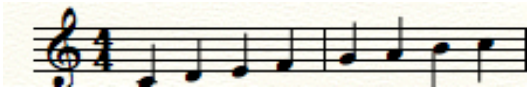
Los eventos más importantes de cara a convertir los archivos MIDI de entrada en matrices de estado son los eventos Nota: ***Note On*** y ***Note Off***. Dichos eventos poseen 4 parámetros:

- ***Tick.*** Instante en el que se ejecuta o se termina la nota.
- **MIDI *Channel*** (de 0 a 16). Canal MIDI en el que se encuentra dicha nota.
- ***Note Number.*** Número de nota de acuerdo a la tabla 3.
- ***Velocity*** (de 0 a 127). Intensidad con la que se toca la nota.

Note On indica el inicio de nota y *Note off* el final de la nota, por tanto, la duración de la nota en *ms* se puede obtener de la diferencia de *ticks* en *Note On* y *Note Off* para una misma nota (nótese que primer valor "x" de data [x,y] en un evento *Note Off* o *Note On* en la figura 10 es igual para una misma nota), por tanto, para notas con un mismo ritmo el tiempo debería ser el mismo (en el ejemplo de la figura 10, dado que el archivo MIDI ha sido generado a partir de un editor de partituras, *Finale*, estos valores no son exactos para figuras iguales).

En **python**, se puede leer un archivo MIDI con la librería `midi` para este lenguaje. Importando la librería mediante la orden `import midi` se puede leer el fichero MIDI, ejecutando: `read_midifile` (nombre_del_archivo.mid), obteniendo la lista de eventos MIDI de dicho archivo. Como ejemplo, para una escala de 8 negras de Do a Do en 4/4, se obtiene la información mostrada en la figura 10.

En un archivo MIDI estándar se puede encontrar información sobre "ticks per quarter note" (PPQ) en los 2 últimos bytes de la cabecera. El PPQ puede tener una resolución de 24-96, aunque se pueden tener resoluciones mayores del orden de 480 o muy altas como 500-1000 que dan el tiempo en milisegundos. Por defecto, para un compás de 4/4 se tiene un tiempo de 120 beats/minuto (bpm), lo que equivaldría a 120 semicorcheas por minuto.



```
midi.Pattern(format=1, resolution=1024, tracks=\
[midi.Track(\
  [midi.SmpteOffsetEvent(tick=0, data=[0, 0, 0, 0, 0]),
   midi.TimeSignatureEvent(tick=0, data=[4, 2, 24, 8]),
   midi.KeySignatureEvent(tick=0, data=[0, 0]),
   midi.SetTempoEvent(tick=0, data=[7, 161, 31]),
   midi.SetTempoEvent(tick=0, data=[7, 161, 35]),
   midi.EndOfTrackEvent(tick=8194, data=[])]),
 midi.Track(\
  [midi.SomethingEvent(tick=0, data=[83, 109, 97, 114, 116, 77, 117, 115, 105, 99, 32, 83,
111, 102, 116, 83, 121, 110, 116, 104, 32, 49]),
   midi.TrackNameEvent(tick=0, text='[Staff 1]', data=[91, 83, 116, 97, 102, 102, 32, 49,
93]),
   midi.ControlChangeEvent(tick=0, channel=0, data=[0, 121]),
   midi.ControlChangeEvent(tick=0, channel=0, data=[32, 0]),
   midi.ProgramChangeEvent(tick=0, channel=0, data=[0]),
   midi.ControlChangeEvent(tick=0, channel=0, data=[7, 101]),
   midi.ControlChangeEvent(tick=0, channel=0, data=[10, 64]),
   midi.ControlChangeEvent(tick=0, channel=0, data=[7, 110]),
   midi.ControlChangeEvent(tick=3, channel=0, data=[100, 0]),
   midi.ControlChangeEvent(tick=0, channel=0, data=[101, 0]),
   midi.ControlChangeEvent(tick=1, channel=0, data=[6, 12]),
   midi.ControlChangeEvent(tick=1, channel=0, data=[38, 0]),
   midi.NoteOnEvent(tick=5, channel=0, data=[60, 69]),
   midi.NoteOffEvent(tick=976, channel=0, data=[60, 0]),
   midi.NoteOnEvent(tick=68, channel=0, data=[62, 65]),
   midi.NoteOffEvent(tick=947, channel=0, data=[62, 0]),
   midi.NoteOnEvent(tick=68, channel=0, data=[64, 67]),
   midi.NoteOffEvent(tick=947, channel=0, data=[64, 0]),
   midi.NoteOnEvent(tick=68, channel=0, data=[65, 65]),
   midi.NoteOffEvent(tick=944, channel=0, data=[65, 0]),
   midi.NoteOnEvent(tick=72, channel=0, data=[67, 69]),
   midi.NoteOffEvent(tick=982, channel=0, data=[67, 0]),
   midi.NoteOnEvent(tick=68, channel=0, data=[69, 65]),
   midi.NoteOffEvent(tick=947, channel=0, data=[69, 0]),
   midi.NoteOnEvent(tick=68, channel=0, data=[71, 67]),
   midi.NoteOffEvent(tick=947, channel=0, data=[71, 0]),
   midi.NoteOnEvent(tick=68, channel=0, data=[72, 65]),
   midi.NoteOffEvent(tick=944, channel=0, data=[72, 0]),
   midi.EndOfTrackEvent(tick=70, data=[])])])
```

Figura 10. Lista de Eventos MIDI para una escala de 8 notas en 4/4 generada a partir de un fichero MIDI cuya partitura ha sido escrita en Finale, y posteriormente leído con la función read_midifile de la librería midi de python que devuelve dicha lista.

Los valores de las notas MIDI o *Note Number* en los eventos *Note On* y *Note Off* se muestran en la tabla 1:

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

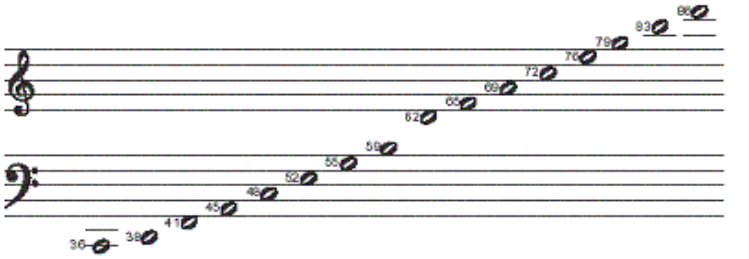


Tabla 1. Números de Nota MIDI

3.3. Análisis de la Red Neuronal

Este trabajo está basado en el software desarrollado por Daniel Johnson [6] que contiene 7 scripts escritos en python que son: data, main, midi_to_statematrix, model, multi_training, out_in_op y visualize. Para que la red funcione, los archivos MIDI de entrada deben estar en una carpeta dentro del directorio de los scripts en python, llamada *music*, y es necesario crear un directorio *output* donde se guardarán los parámetros de entrenamiento y las salidas MIDI en cada epoch. Además, se deberá incluir un fichero que se denominará *parameters*, que cargue el número de neuronas por capa y haga referencia a la carpeta donde se guardarán los archivos resultantes. A continuación se muestra un diagrama de los ficheros que contiene la red.

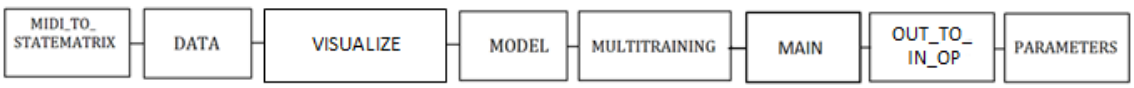


Figura 11. Diagrama de los ficheros que forman la red

El fichero **midi_to_statematrix**.py se encarga de transformar las obras de entrada en formato MIDI a una matriz de estados que se describe a continuación, que se transforma en el vector de entrada en el fichero **data**.py del que se obtendrá la entrada final de la red. Las piezas son cargadas mediante la función `loadPieces` del fichero multi_training.py. En este fichero se definen 3 parámetros:

- ***Batch_width***. Número de secuencias en cada *batch*, es decir, el número de obras que se entrenan a la vez en la red. Nótese que no pueden entrenarse todas las obras de la base de datos a la vez debido al tamaño de la misma y el coste computacional que supondría.
- ***Batch_len*** (x*y). Tamaño de cada secuencia multiplicado por el número de compases a entrenar de cada obra.
- ***Division_len***. Valor de la subdivisión por compás a entrenar, en este caso 16, que equivale a la figura de semicorchea.

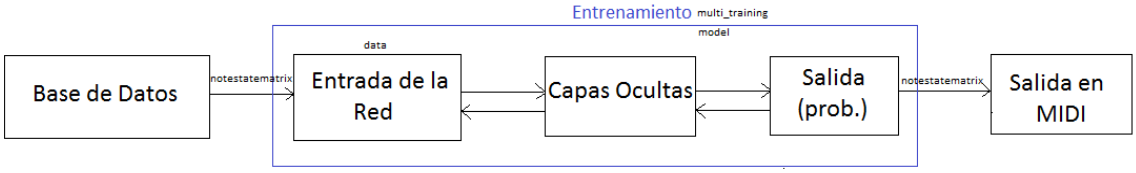


Figura 12. Diagrama de bloques del algoritmo

Se obtienen los segmentos de las piezas que serán transformadas en el vector de entrada y se implementa el algoritmo de entrenamiento mediante la función *trainPiece*.

Por otro lado, el fichero **model**.py contiene el modelo de la red neuronal recurrente que junto con **visualize**.py y **out_to_in_op**.py forman la estructura de la red neuronal. Las notas de salida serán calculadas a través de dos probabilidades descritas en el sexto apartado de este capítulo.

Finalmente, en el fichero **parameters**.py creado se introduce el número de neuronas en cada una de las 4 capas con el que se desee entrenar la red y el tamaño del fichero .mid de salida con un número multiplicador del número de compases de entrada definidos en *batch_len*. Para generar más piezas a partir de los parámetros de entrenamiento, se escribió un script que toma el fichero final_learned_config.p en la carpeta *output* que contiene dichos parámetros y genera otra obra distinta a la generada tras el entrenamiento de la red, para no tener que entrenar de nuevo la red. Esto permite la generación de múltiples obras distintas basadas en un mismo entrenamiento.

3.4. Entrada de la Red

Al introducir las obras de entrada, una función de un fichero en python se encarga de comprobar si el formato de las obras es el adecuado (MIDI) y el compás 4/4. Una vez cargadas las obras, la red se queda con tantos compases como se especifique por defecto en el parámetro *batch_len*. Esto se hace para evitar un exceso de información que haría el entrenamiento mucho más lento e innecesario, ya que como se ha descrito antes, la forma sonata se compone a partir de pequeños fragmentos musicales que son desarrollados a lo largo de la obra.

3.4.1. Matriz de Estados

Una vez se obtienen los fragmentos de todas las obras, mediante un parser MIDI escrito en python se transforman las obras en una matriz de estados o *statematrix*. Esta matriz tiene como dimensiones 79 columnas, y filas el número de semicorcheas por compás más una última fila en blanco, es decir, para un compás de 4 negras el número de filas sería 4x4 semicorcheas/negra = 8 filas más la última fila de ceros.

La matriz de estados sólo da información de la situación de la nota interpretada o tocada en un instante de tiempo. Dicha matriz se muestra en la expresión 7:

$$\begin{bmatrix} [0,0] & \dots & [0,0] \\ \vdots & \ddots & \vdots \\ \dots & \dots & [0,0] \end{bmatrix} \quad (7)$$

El motivo por el que esta matriz tiene 78 columnas (más una adicional de ceros) es porque en el parser MIDI se definen dos cotas de notas, una inferior igual a 24 y otra superior igual a 102, siendo en la matriz la primera columna igual al valor de nota 24 y la última a 102. La diferencia de estas cotas, 78, es el número de columnas. Para un Do de valor MIDI 60, en la columna 37 (23+37 = 60 ya que la nota 24 será la columna 1) aparecerá un valor distinto de cero.

Cada elemento de la matriz posee 2 valores que por defecto son 0:

- El **1er valor** es el "**play**" y hace referencia a si la nota está siendo tocada en ese instante de tiempo correspondiente a la fila actual (o semicorchea actual) en cuyo caso será 1, y 0 si no es esa nota la que está sonando.

- El **2º valor** es el "**articulate**" que corresponde a la articulación de la nota. La nota, por ejemplo una negra, en su primer evento *Note* (será *Note On*) tendrá un valor 1, y en los 3 eventos o filas siguientes el valor será 0, ya que la nota ya ha sido articulada (como si se pulsara en el piano una tecla, la nota al principio sería tocada y articulada [1,1] pero en los siguientes instantes de tiempo seguiría tocada pero no articulada ya que la tecla seguiría presionada [1,0]).

Un ejemplo de parte de esta matriz de estados (de la columna 28 a la 49) para la escala de la figura 10, será la mostrada en la figura 13.

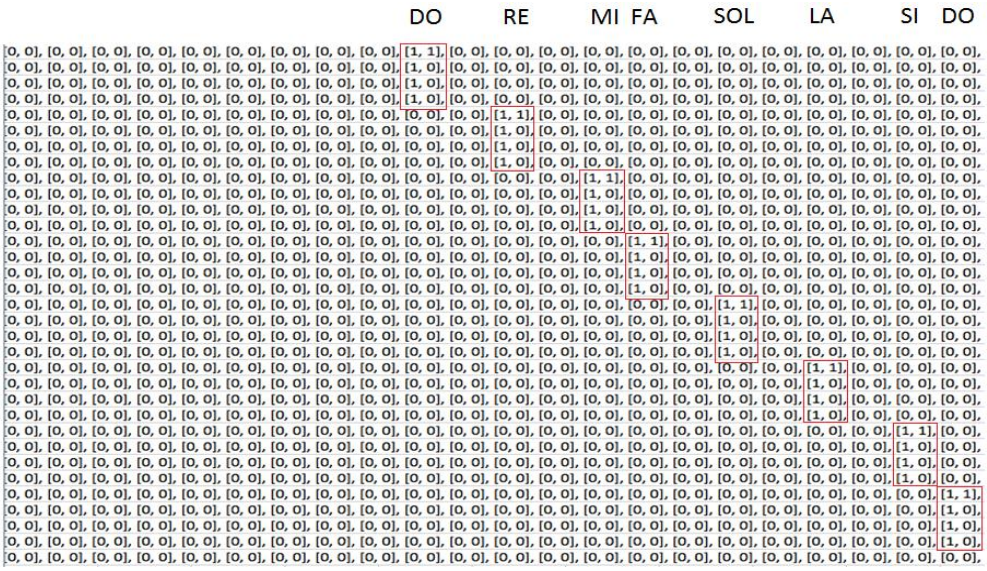


Figura 13. Matriz de estados para la escala de la figura 11

En la figura anterior se puede observar en la columna 37 el inicio del Do (60) en unísono (al ser negra tendrá 4 filas), luego un Re en la columna 39 filas 5-8 (nota 62 en la tabla 3), y así hasta llegar al Do final (72) en las filas 13-16 para terminar con una fila final de ceros.

3.4.2. Matriz de Entrada

Una vez obtenida la matriz de estados, se calcula en el script data.py la matriz o vector de entrada de la red. Esta matriz posee el mismo número de filas que la matriz de estados y 78 columnas. Cada columna posee a su vez 80 elementos ordenados por: posición, pitchclass, vecindad anterior, contexto anterior y beat (el número entre llaves muestra el número de cifras de cada elemento en el vector de longitud 80). El último elemento que se encuentra después del beat es un 0.

- **Posición** {1}. Posición de la nota de 0 a 77 que equivale respectivamente de 24 a 102 en notas MIDI según la tabla 3.
- **Pitchclass** {12}. Muestra la posición de la nota en la escala musical de 12 notas. Los 12 elementos son 0 salvo en la posición de la nota que es un 1.
- **Vecindad Anterior** {50}. Dividido en 2 tramos de 25 elementos cada uno, para cada tramo hay 12 elementos "play" y otros 12 "articulate" que son 0 o 1 dependiendo si la nota actual ha sido tocada y/o articulada en el evento de tiempo anterior. Los elementos 25 y 50 son 0.
- **Contexto Anterior** {12}. Con la misma filosofía que el pitchclass pero en este caso dando información de la nota actual respecto de las que están sonando.
- **Beat** {4}. Representación binaria de la posición de la nota en el compás acotada de -1 a 1, que sigue el patrón:

0101010101010101
0011001100110011
0000111100001111
0000000011111111

3.5. Estructura de la Red

La estructura de la red neuronal utilizada comprende un tipo de red biaxial: LSTM para el tiempo en el eje z, y otra red LSTM en el eje y para las notas. En la figura x se puede observar la estructura final de la red. Las dos primeras capas tienen conexiones en el eje del tiempo, pero no en el de las notas. Las últimas dos capas tienen conexiones en el eje de las notas pero no en el eje del tiempo.

Cada fila de la matriz de entrada es una entrada en el eje de notas y, por tanto, para una sola entrada de un compás de 4 negras se tendrán 17 filas en la matriz de entrada (la última de ceros), es decir, 17 entradas (cuadrados amarillos en la figura 14) en el eje y. El eje x es la dirección de computación.

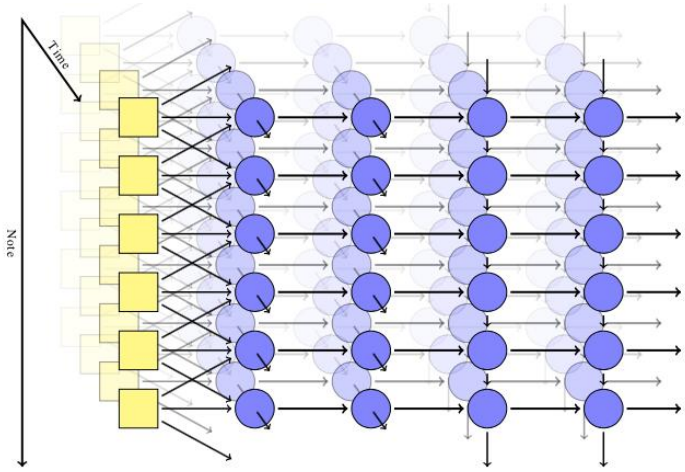


Figura 14. Estructura final de la red neuronal [6]

3.6. Salida

Una vez aplicado el algoritmo Backpropagation y ajustados los pesos de la red, se obtienen como salida dos probabilidades (que es el tamaño de la última capa de la red) para cada nota dentro del rango definido en el parser MIDI. La primera nota del fichero de salida será la primera nota de uno de los ficheros de entrada escogida al azar, y a partir de ahí se colocarán las demás.

- **Play Probability.** Probabilidad de la nota de ser escogida para ser tocada.
- **Articulate Probability.** Probabilidad de articulación de la nota, una vez que es tocada. La articulación de la nota se refiere a, si se tienen dos notas iguales consecutivas, tocar cada una por separado en lugar de ligarlas o mantenerlas.

Por ejemplo, si las probabilidades de la nota Do (C) son [0.7,0.3], el fichero de salida incluirá dicha nota un 70% del tiempo con una repetitividad del 30%, es decir, sonará dos veces esa nota seguida con una probabilidad del 30%.

3.7. Interfaz Gráfica

Una vez comprendida la estructura y la entrada de la red, dado que es necesario tener conocimientos de python para generar música, se ha decidido crear una interfaz gráfica que permita un uso más sencillo y visual de las funcionalidades de la red. Para ello, se ha escrito un script en python utilizando su módulo Tkinter, ya que es un módulo que no requiere instalación, pues viene integrado en python, y es multiplataforma.

Para la realización de la interfaz se importan todos los módulos necesarios y a continuación, se crea una ventana de inicio *root*, con sus dimensiones y título. Además se sitúa una etiqueta con texto de introducción al software.

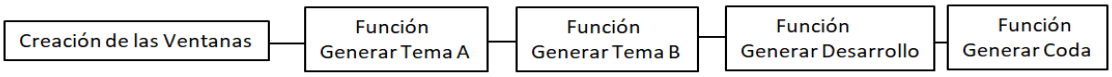


Figura 15. Diagrama de bloques del código de la interfaz gráfica

En esta ventana de inicio se crean además 3 botones correspondientes cada uno a un compositor, en este caso Mozart, Beethoven y Schubert, que son los compositores cuyas obras se han utilizado para entrenar la red, y un botón de *close* para terminar el proceso.

A continuación, para crear la orden para que al seleccionar un botón aparezca una ventana nueva con el nombre, foto, fecha de nacimiento y muerte del compositor, y 5 botones para generar música entre los que se encuentran Tema A, Tema B, Coda, Desarrollo y Sonata Completa (suma de los 4 botones anteriores), y un botón *close* para cerrar la ventana y volver a la ventana principal. La funcionalidad de cada uno de esos botones para generar música vendrá definida por otra función.

Cada compositor tiene una ventana igual donde se podrá generar música a partir de un entrenamiento de obras de todos los compositores. La interfaz final se muestra en la figuras 16, 17, 18 y 19.

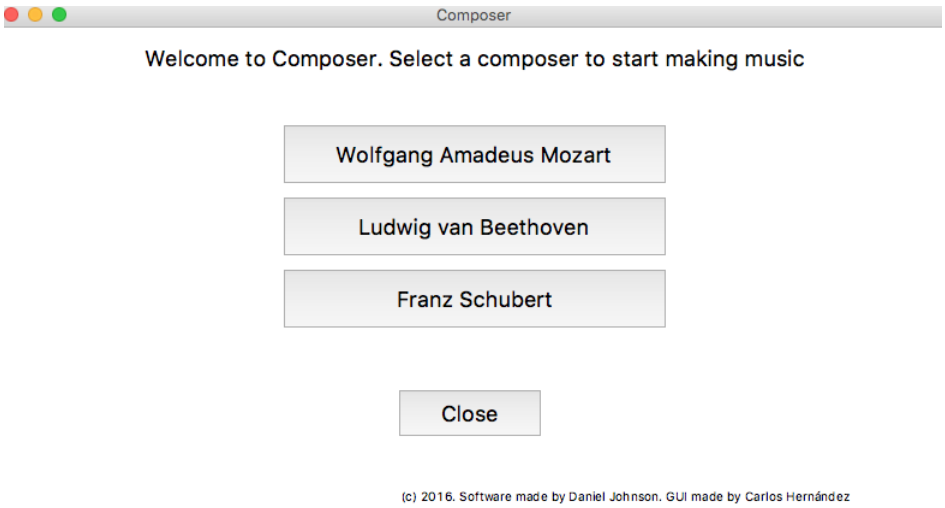


Figura 16. Ventana principal interfaz gráfica

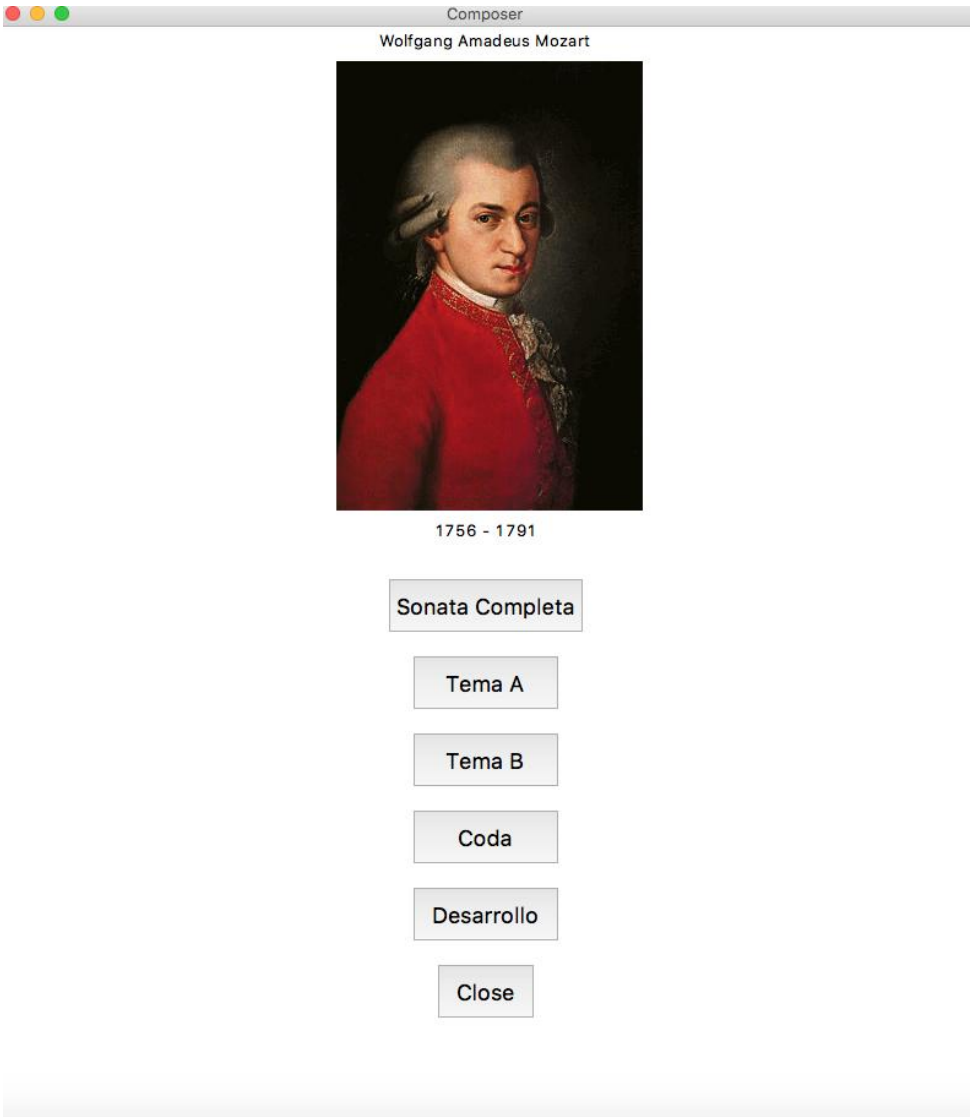


Figura 17. Ventana Mozart interfaz gráfica

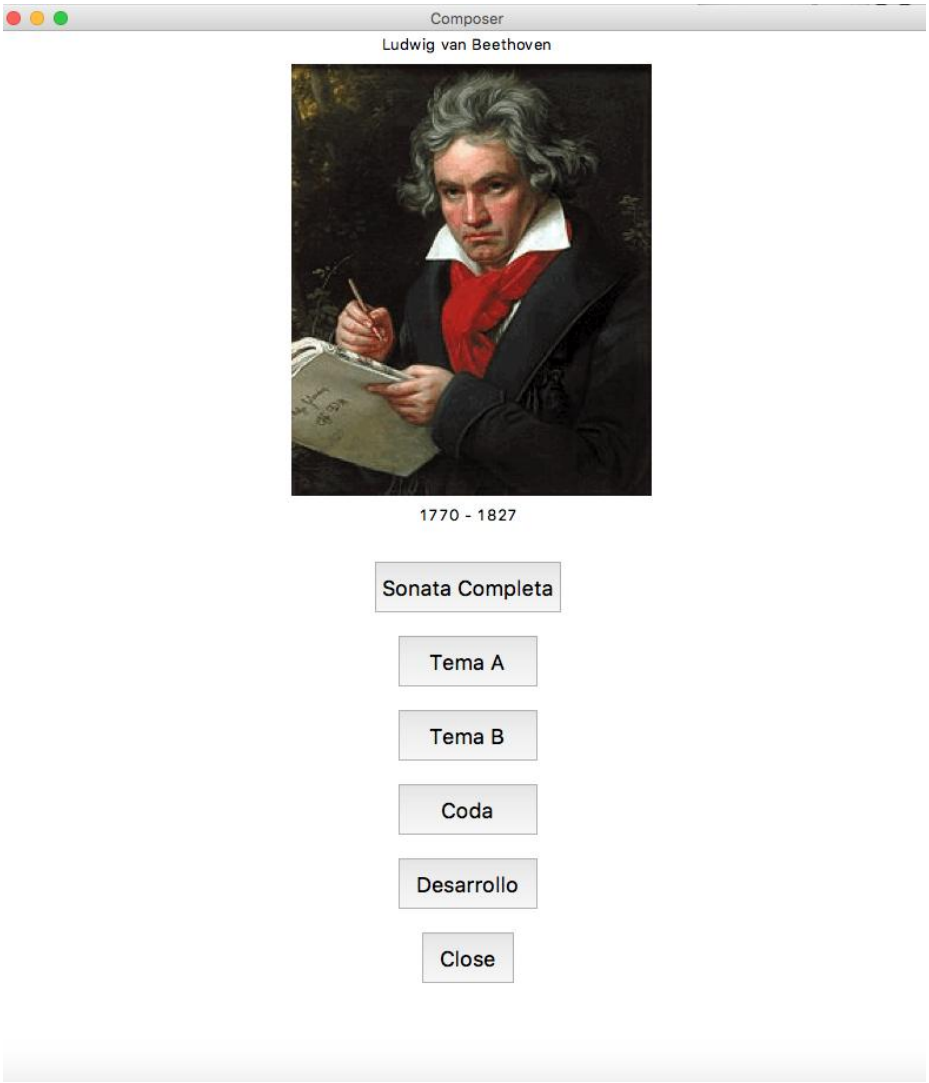


Figura 18. Ventana Beethoven interfaz gráfica

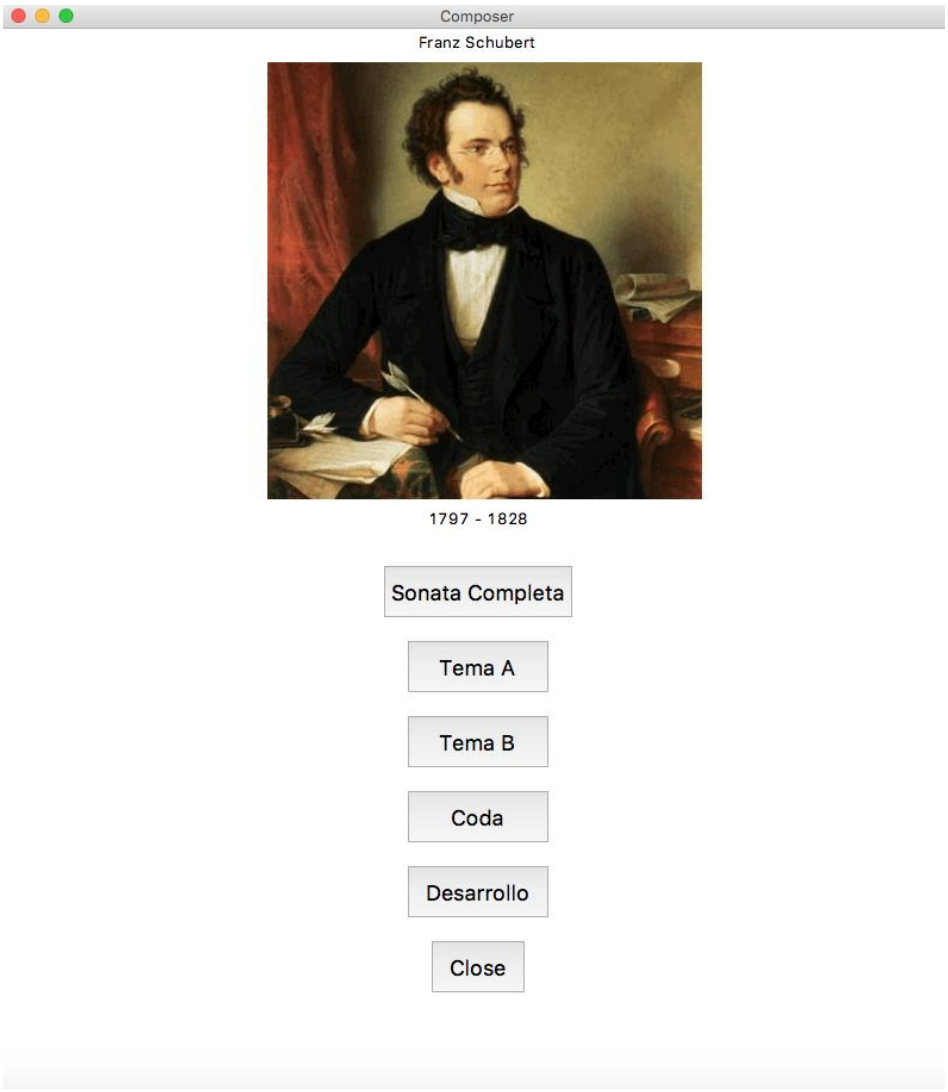


Figura 19. Ventana Schubert interfaz gráfica

Capítulo 4

Análisis de los Resultados

En este capítulo se exponen y recogen todos los resultados obtenidos tras el entrenamiento de la red neuronal. Comenzando por el primer entrenamiento del software original, se exponen los resultados y planes de acción seguidos tras el análisis de los resultados para el cálculo del número óptimo de neuronas y epochs de la red.

4.1. Entrenamiento con el Software Original

Tal y como está definido el código de la red, se entrenan fragmentos de 8 compases de obras en compás de 4/4. Para ello, se deben introducir solamente obras en 4/4 en la carpeta *music* y, tras el entrenamiento se obtendrá una obra en la carpeta *output* en formato MIDI. En cuanto al número de neuronas, se disponen de 300 neuronas en cada una de las dos primeras capas ocultas, y 100 y 50 en las dos siguientes respectivamente. El entrenamiento dura hasta 10.000 *epochs*.

Dado que el código original sólo admite obras en 4/4, se introdujeron en el código unas líneas para que el software no se detuviera al encontrar una obra en un compás diferente a 4/4, sino que pasase a evaluar la siguiente obra.

Al comienzo, se tomó la red neuronal diseñada por Daniel Johnson [6] ya descrita en el capítulo anterior. El primer entrenamiento se realizó en un equipo iMac de 2,7 Ghz intelCore i5 con una memoria de 8 GB. Las entradas a la red fueron Sonatas para piano de Mozart, ya que al ser obras para piano se disminuía desde un primer momento la dificultad de tener múltiples voces.

Los resultados obtenidos se muestran en la figura 20. Aunque el error disminuye notablemente desde la primera *epoch* hasta la última, musicalmente los resultados no son concluyentes. El entrenamiento no resulta en un aprendizaje de conceptos tales como la tonalidad o la estructura de las obras.

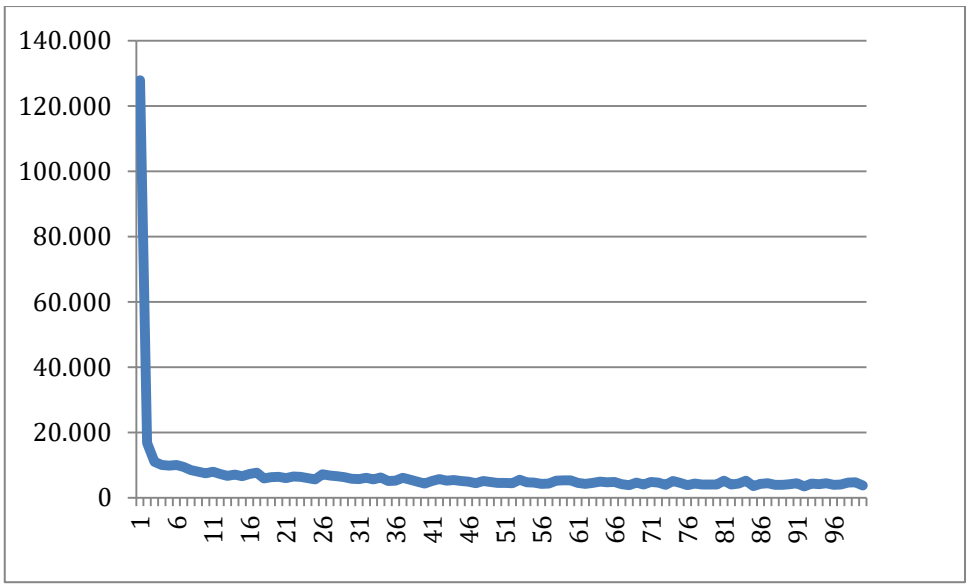


Figura 20. Error en el entrenamiento con Sonatas de Mozart

A partir de este momento, dado que la red podía entrenarse y las salidas obtenidas se decidió modificar el código de la red neuronal para que, a partir de Sonatas de Mozart para piano la salida fueran obras en Forma Sonata, en definitiva, hacer una red que compusiera movimientos en Forma Sonata.

4.2. Análisis de los Resultados de Entrenamiento

Dado que el autor del software no especifica los motivos de la elección del número de neuronas ni *epochs* empleadas en el entrenamiento, se procede a realizar un análisis de las mismas buscando el número óptimo de estos dos parámetros para conseguir los mejores resultados posibles. Este análisis se realizará sobre los Temas A (se recomienda ver el Anexo II para la comprensión de la estructura de estos temas). A continuación se mostrarán los entrenamientos llevados a cabo con sus respectivos errores graficados mediante la librería *matplotlib* de python, y la evaluación de los resultados siguiendo los parámetros de la tabla 2 valorándolos en una escala del 1 (escaso) al 5 (bueno), aunque el test definitivo para evaluar los resultados es el *Test de Turing* [12].

Melodía	Armonía	Originalidad	Forma	Textura

Tabla 2. Parámetros de evaluación de los resultados de entrenamiento

Los parámetros serán evaluados con un 1 si: en el caso de la **melodía** el ámbito es escaso, el registro estático, etc, la **armonía** no posee cadencias o complejidad armónica, la **originalidad** si el resultado es una copia de cualquiera de las entradas, la **forma** si la melodía no sigue un patrón rítmico marcado por el motivo (ver Anexo II), y la **textura** si es monódica. En cuanto a este último parámetro, se evaluará con un 3 si es melodía acompañada y 5 si es polifónica.

El código adicional introducido en el fichero multi_training.py para hacer una gráfica del error frente a las *epochs* con la librería matplotlib es el siguiente:

```
import matplotlib.pyplot as plt
plt.plot(i, error, 'bo', linewidth=2.0)
plt.xlabel('EPOCH')  #Etiqueta eje x
plt.ylabel('ERROR')  #Etiqueta eje y
plt.xlim(-100, 10000)  #Limites eje x
plt.ylim(-200, 50000)  #Limites eje y
plt.minorticks_on()
plt.savefig('Error Entrenamiento.pdf')
```

4.2.1. Entrenamiento variando el Número de Epochs

Para comenzar el estudio de los entrenamientos, se han fijado el número de neuronas a 300, dado que es el máximo número de neuronas que utiliza el autor en una capa. Una vez fijado este número de neuronas en las 4 capas ocultas se variará el número de *epochs* y se valorarán los resultados obtenidos escogiendo el mejor entrenamiento realizado.

4.2.1.1. Entrenamiento con 2000 Epochs

Antes de comenzar el primer entrenamiento se obtuvieron las obras de entrada (sonatas) en formato MIDI de los autores Mozart, Beethoven y Schubert. Tras analizar cada una de las sonatas se separaron los temas A, temas B, codas y desarrollos de cada obra con el programa *Finale*. Una vez obtenidos los fragmentos de cada obra se realiza

el primer entrenamiento estableciendo 300 neuronas en cada una de las 4 capas, 2000 epochs y 4 compases de entrada por obra. La salida será 5 veces el valor de la entrada, por tanto, 20 compases. El resultado transcrito a partitura se muestra en la figura 21 y los errores obtenidos, en la figura 22. La tabla 3 muestra la valoración de la obra obtenida tras el entrenamiento. Este entrenamiento tuvo una duración de 12h.

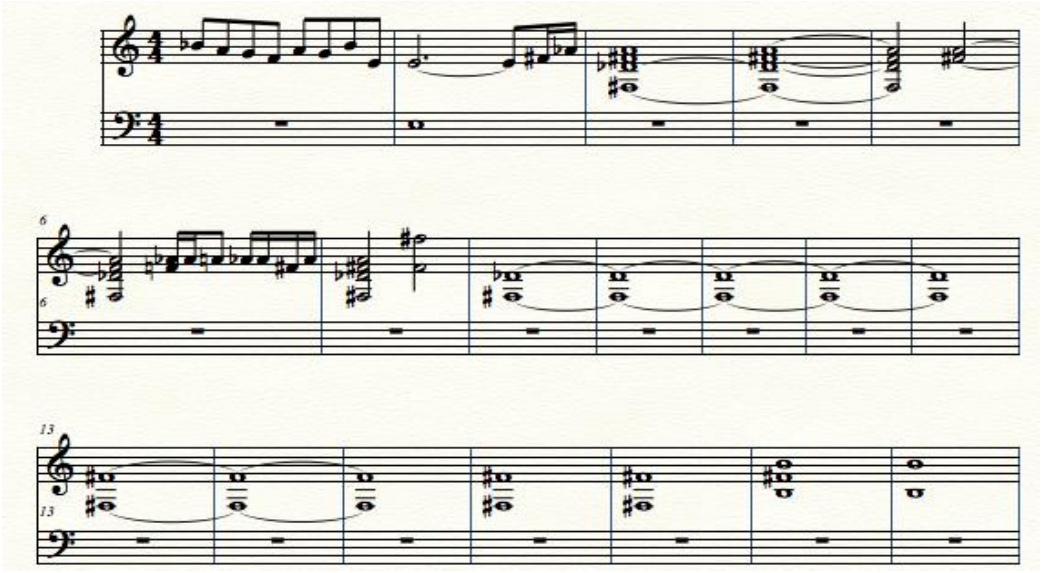


Figura 21. Resultado entrenamiento Temas A con 300 neuronas por capa y 2000 epochs

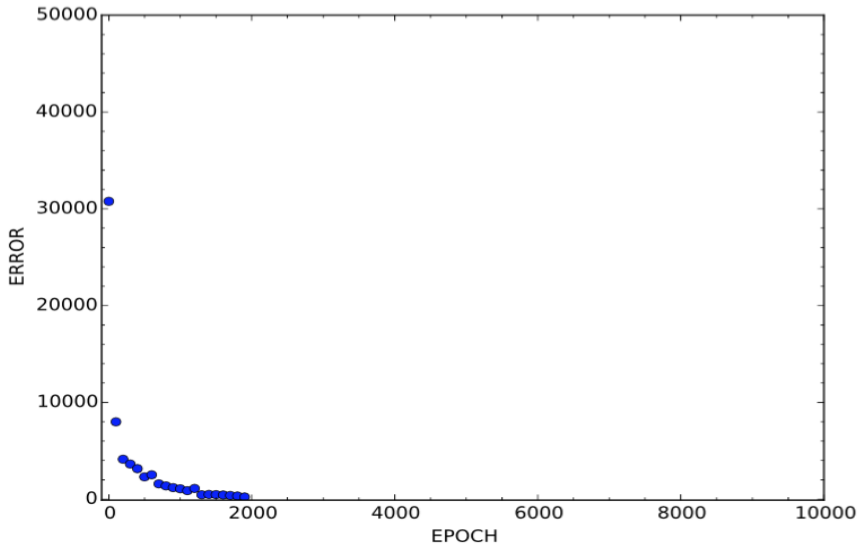


Figura 22. Error entrenamiento Temas A con 300 neuronas por capa y 2000 epochs

Melodía	Armonía	Originalidad	Forma	Textura
1	2	2 (1er compás copia de Mozart)	1	2

Tabla 3. Parámetros de evaluación de los resultados de entrenamiento de Temas A con 300 neuronas por capa y 2000 epochs

4.2.1.2. Entrenamiento con 5000 Epochs

Tras el análisis de los resultados obtenidos en el primer entrenamiento se decidió entrenar la red por segunda vez, esta vez variando únicamente el número de *epochs* respecto del entrenamiento anterior. Este entrenamiento tuvo una duración de un día.

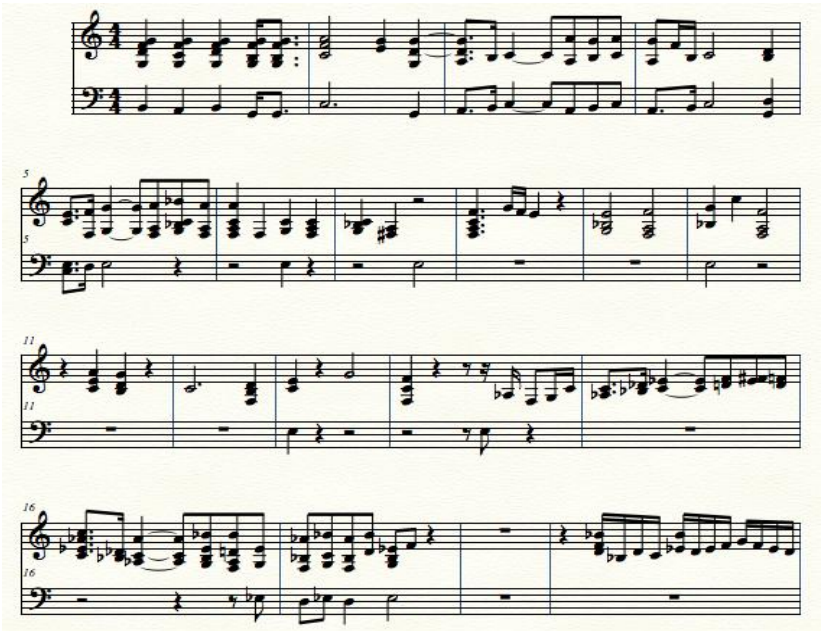


Figura 23. Resultado entrenamiento Temas A con 300 neuronas por capa y 5000 epochs

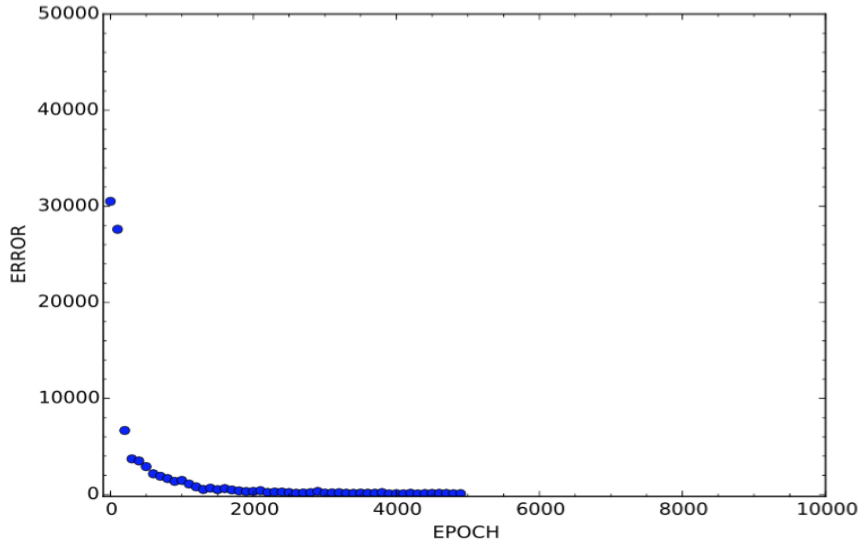


Figura 24. Error entrenamiento Temas A con 300 neuronas por capa y 5000 epochs

Melodía	Armonía	Originalidad	Forma	Textura
4	4	5	2	3

Tabla 4. Parámetros de evaluación de los resultados de entrenamiento de Temas A con 300 neuronas por capa y 5000 epochs

4.2.1.3. Entrenamiento con 10000 Epochs

A pesar de que los resultados del segundo entrenamiento son muy buenos, se realizó un tercer entrenamiento aumentando hasta 10000 el número de *epochs*, como cita el autor [6]. El objetivo de este entrenamiento es, tras los resultados obtenidos anteriormente, el de conseguir mejorar aspectos como la forma o textura de la pieza.

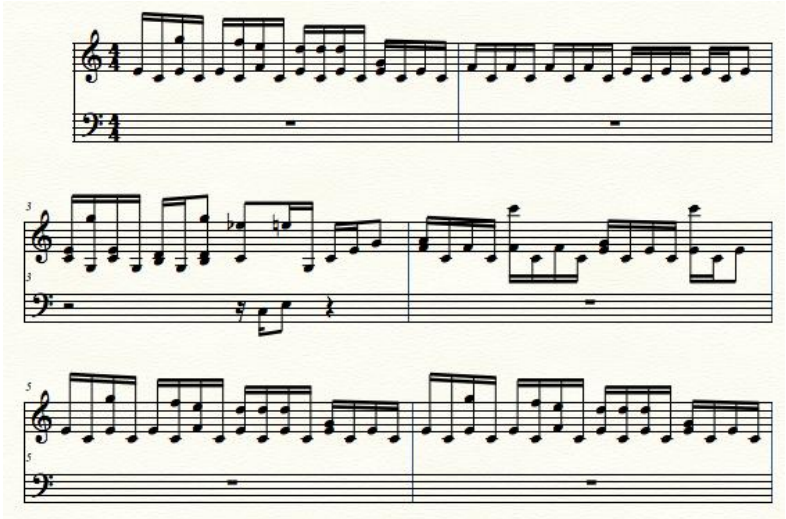


Figura 25. Resultado entrenamiento Temas A con 300 neuronas por capa y 10000 epochs

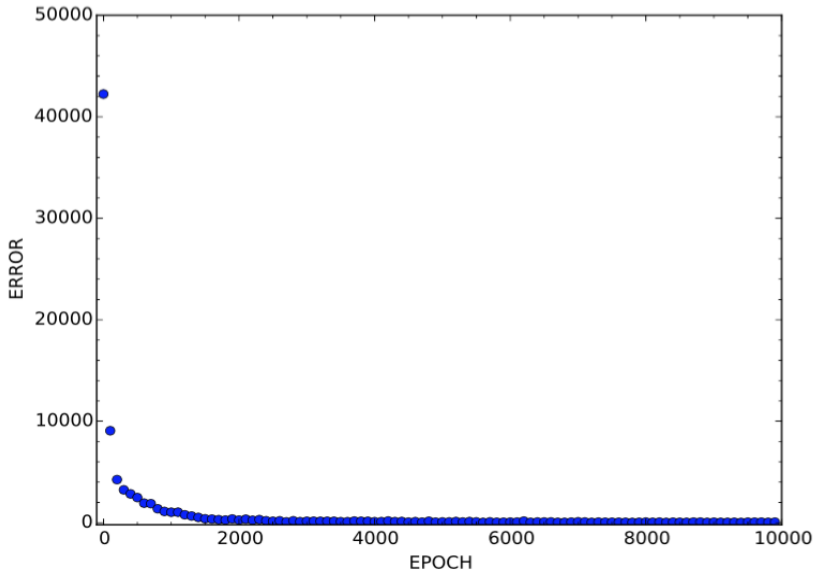


Figura 26. Error entrenamiento Temas A con 300 neuronas por capa y 10000 epochs

Melodía	Armonía	Originalidad	Forma	Textura
5	5	1 (copia íntegra de una sonata de Mozart)	5	5

Tabla 5. Parámetros de evaluación de los resultados de entrenamiento de Temas A con 300 neuronas por capa y 10000 epochs

Dado que la obra es prácticamente una copia de una sonata de Mozart no se puede tomar este resultado como bueno. Obviamente los otros parámetros son musicalmente

muy buenos, pero el hecho no generalizar ya que se obtiene una salida copia de una entrada hace que este entrenamiento no sea válido.

Por tanto, se concluye que el número de neuronas por red oculta debe ser 300 y el número de *epochs* 5000, que son los parámetros establecidos en el segundo entrenamiento.

4.2.2. Entrenamientos variando el Número de Neuronas

Una vez establecido el número de *epochs* ideal, se va a variar el número de neuronas en las capas ocultas. Fijando el número de *epochs* en 5000, se realizará un entrenamiento con 100 neuronas y otro con 500.

4.2.2.1. Entrenamiento con 100 Neuronas por capa

Variando el número de neuronas se podrá determinar el entrenamiento óptimo, ya que el número de *epochs* está definido. El entrenamiento con 100 neuronas tuvo una duración de apenas 8 horas, lo que a priori hace que sea mucho menos costoso computacionalmente que los anteriores.

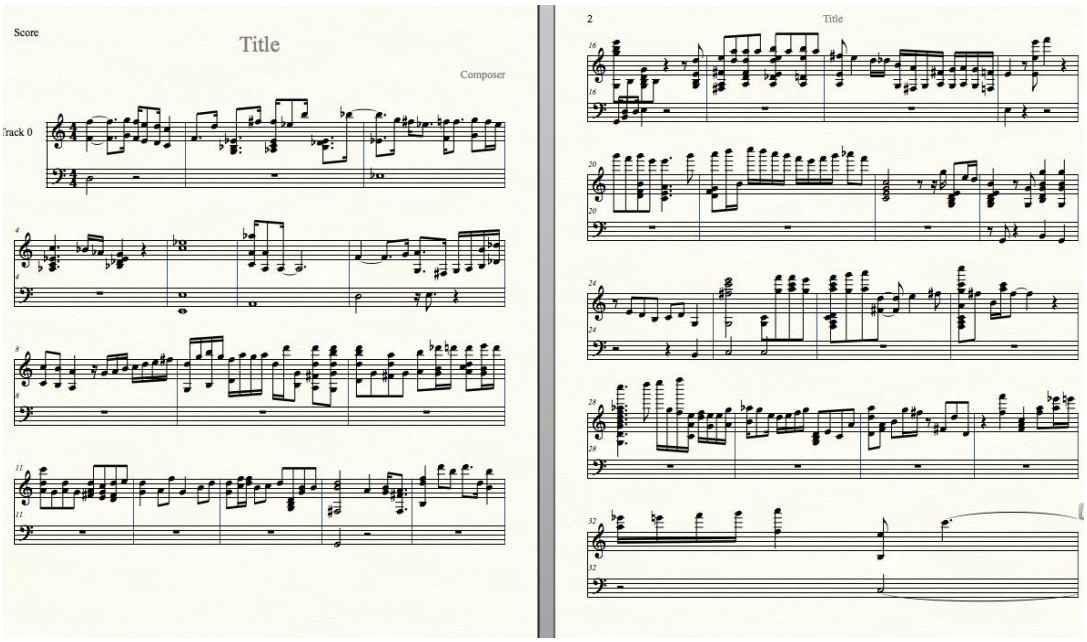


Figura 27. Resultado entrenamiento Temas A con 100 neuronas por capa y 5000 epochs

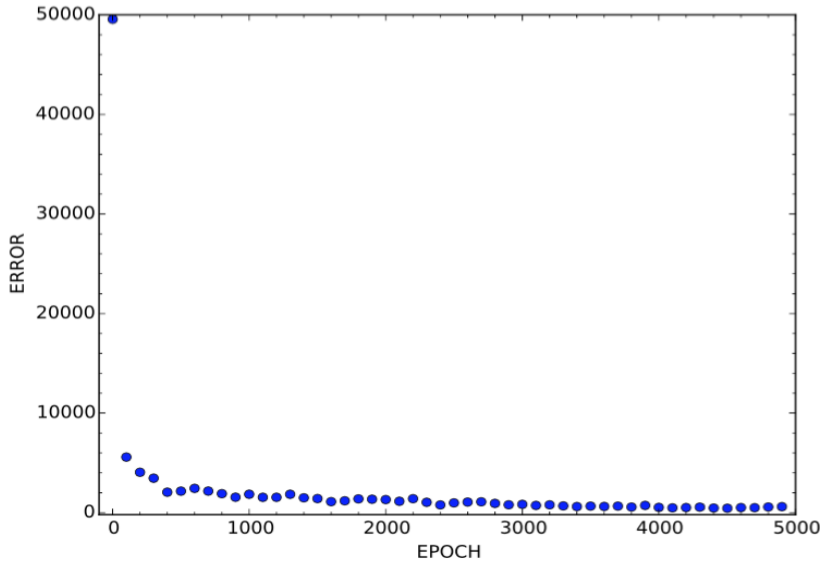


Figura 28. Resultado entrenamiento Temas A con 100 neuronas por capa y 5000 epochs

Melodía	Armonía	Originalidad	Forma	Textura
5	4	5	5	5

Tabla 6. Parámetros de evaluación de los resultados de entrenamiento de Temas A con 100 neuronas por capa y 5000 epochs

Sin haber realizado un entrenamiento superior a 300 neuronas todavía, este entrenamiento resulta mejor que todos los anteriores.

4.2.2.2. Entrenamiento con 500 Neuronas por capa

Al realizar el entrenamiento con 500 neuronas por capa y 5000 *epochs*, el coste computacional es demasiado alto. En un día, la red no consiguió llegar ni a la *epoch* 100 que en los entrenamientos anteriores había tardado como mucho dos horas, por tanto, se decidió detener el entrenamiento.

Analizando no los resultados de entrenamiento, sino las obras generadas a partir de los mismos, se observó que en las obras generadas a partir del entrenamiento con 300 neuronas y 5000 *epochs* tenían cierta tendencia a copiar las entradas, cosa que no pasa en el entrenamiento con 100 neuronas, por tanto, se concluye que el número óptimo de neuronas es 100 y de *epochs* 5000.

Nótese que se podría variar el número de compases entrenados por obras de 4 a 8, o la subdivisión por compás de 16 (semicorcheas) a 32 (fusas), lo que podrían hacer que los resultados de los entrenamientos fueran mejores, pero a cambio de costes computacionales bastante más altos imposibles de afrontar con los equipos con los que se ha desarrollado este trabajo.

A continuación, dado que el entrenamiento óptimo se tiene con un número de 100 neuronas y 5000 epochs, se van a comparar los resultados de entrenamiento para Temas

B, Codas y Desarrollos con 100 y 300 neuronas y el mismo número de epochs, 5000, para verificarlo.

4.2.3. Resultados del Entrenamiento Final para Temas B

- Resultado de entrenamiento con 300 neuronas y 5000 *epochs*:

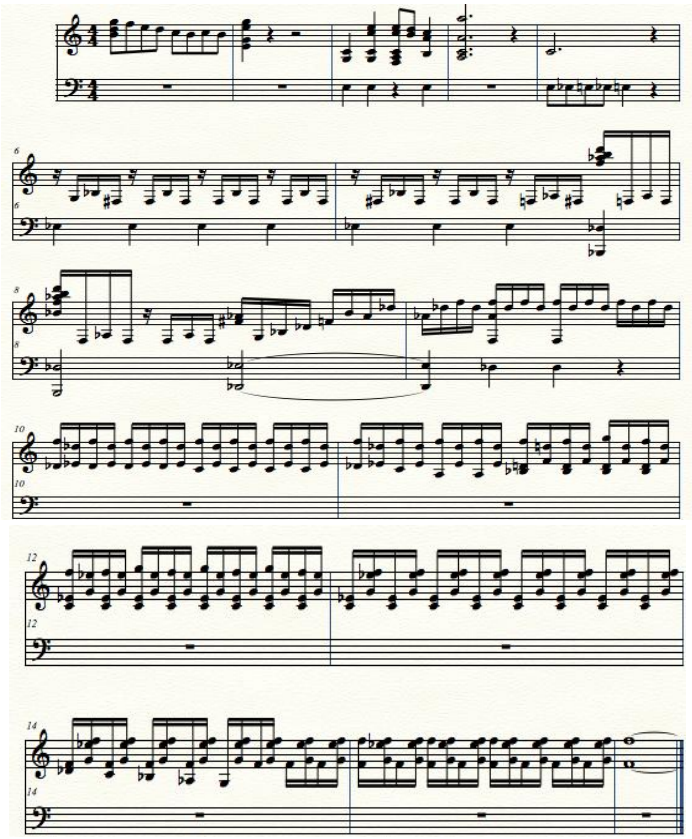


Figura 29. Resultado entrenamiento Temas B con 300 neuronas por capa y 5000 epochs

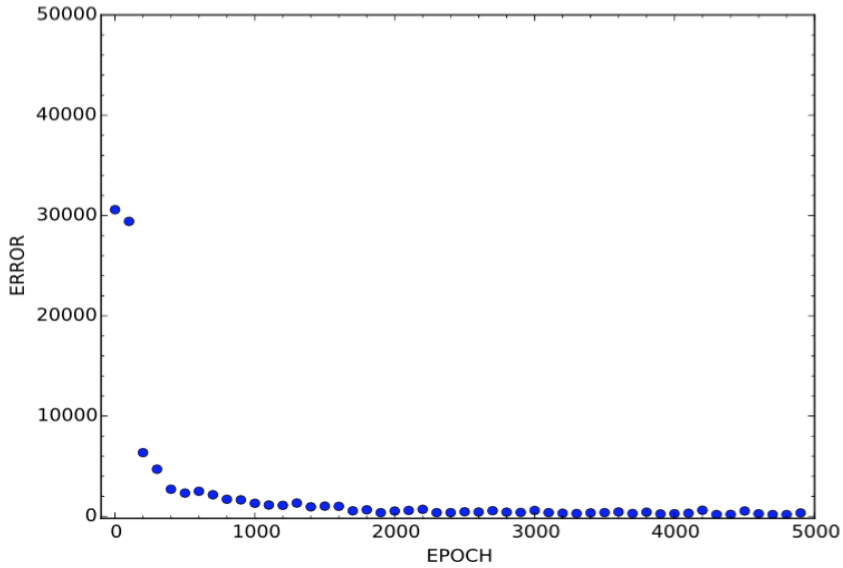


Figura 30. Error entrenamiento Temas B con 300 neuronas por capa y 5000 epochs

Melodía	Armonía	Originalidad	Forma	Textura
3	4	5	4	5

Tabla 7. Parámetros de evaluación de los resultados de entrenamiento de Temas B con 300 neuronas por capa y 5000 epochs

- Resultado de entrenamiento con 100 neuronas y 5000 epochs:

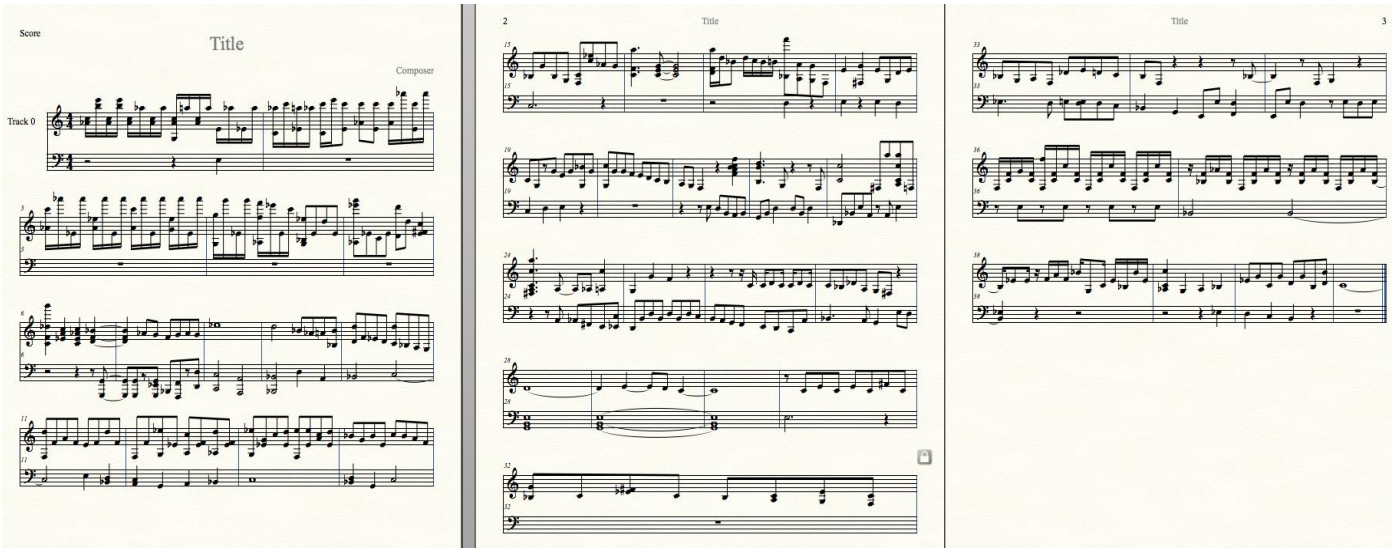


Figura 31. Resultado entrenamiento Temas B con 100 neuronas por capa y 5000 epochs

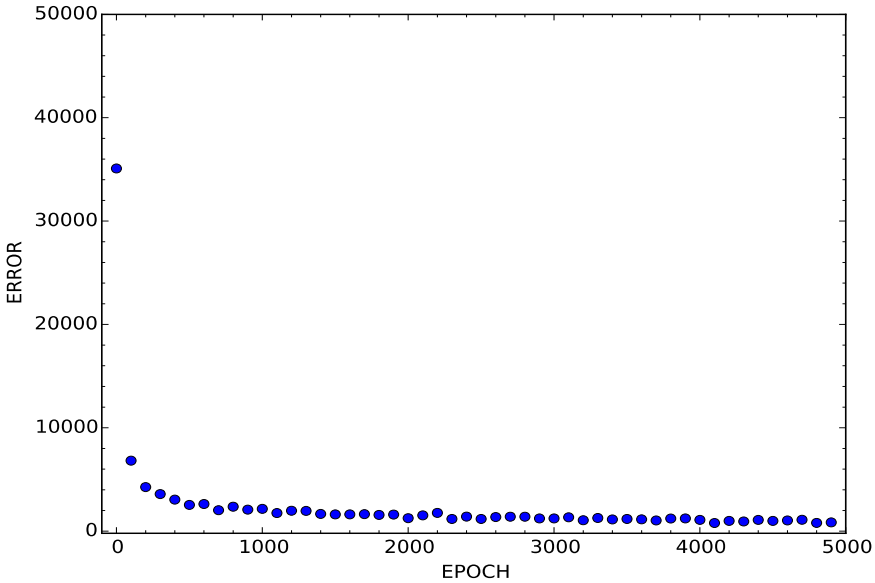


Figura 32. Error entrenamiento Temas B con 100 neuronas por capa y 5000 epochs

Melodía	Armonía	Originalidad	Forma	Textura
5	5	5	5	5

Tabla 8. Parámetros de evaluación de los resultados de entrenamiento de Temas B con 100 neuronas por capa y 5000 epochs

4.2.4. Resultados del Entrenamiento Final para Codas

- Resultado de entrenamiento con 300 neuronas y 5000 epochs:

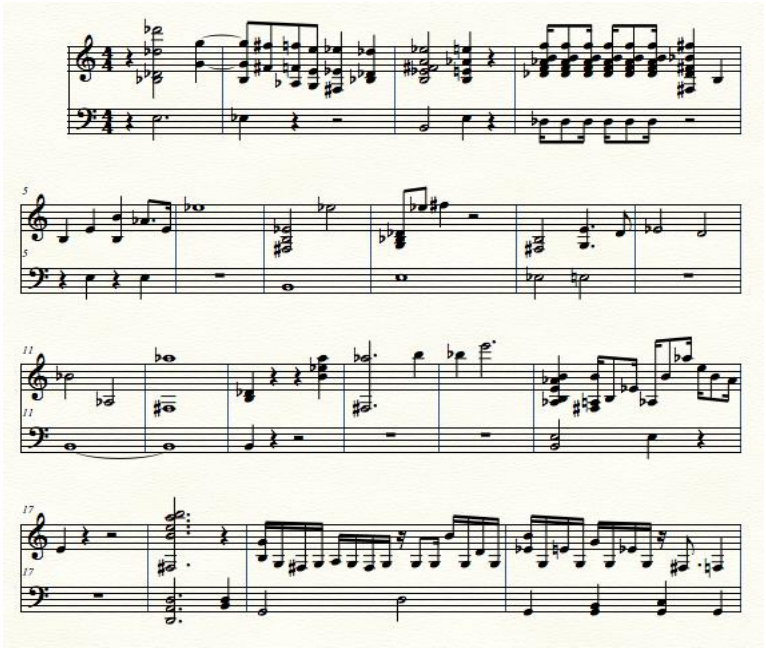


Figura 33. Resultado entrenamiento Codas con 300 neuronas por capa y 5000 epochs

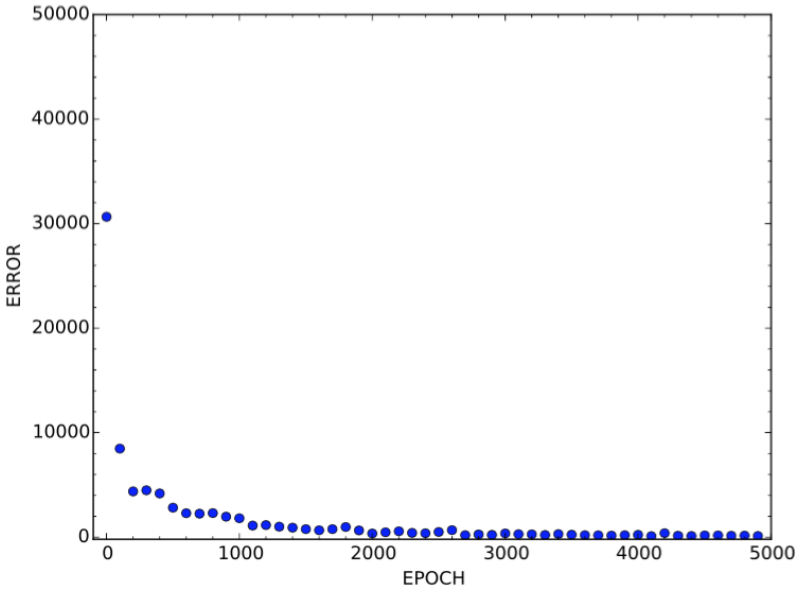


Figura 34. Error entrenamiento Codas con 300 neuronas por capa y 5000 epochs

Melodía	Armonía	Originalidad	Forma	Textura
3	3	5	3	4

Tabla 9. Parámetros de evaluación de los resultados de entrenamiento de Codas con 300 neuronas por capa y 5000 epochs

- Resultado de entrenamiento con 100 neuronas y 5000 epochs:

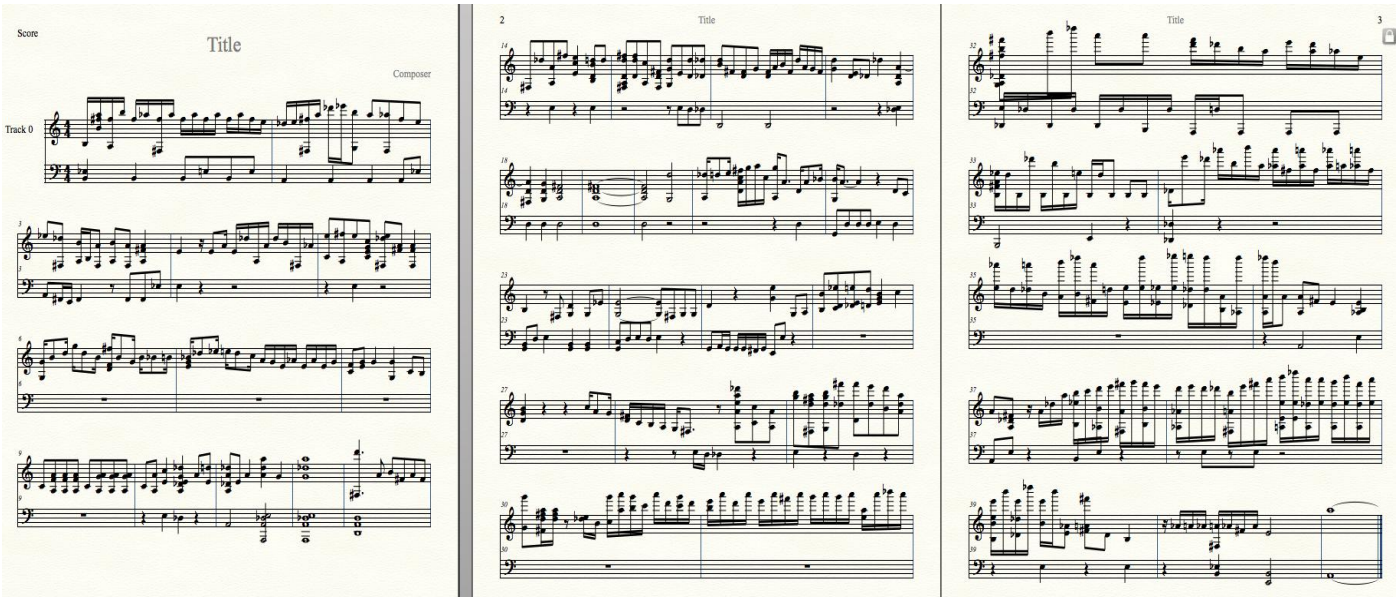


Figura 35. Resultado entrenamiento Codas con 100 neuronas por capa y 5000 epochs

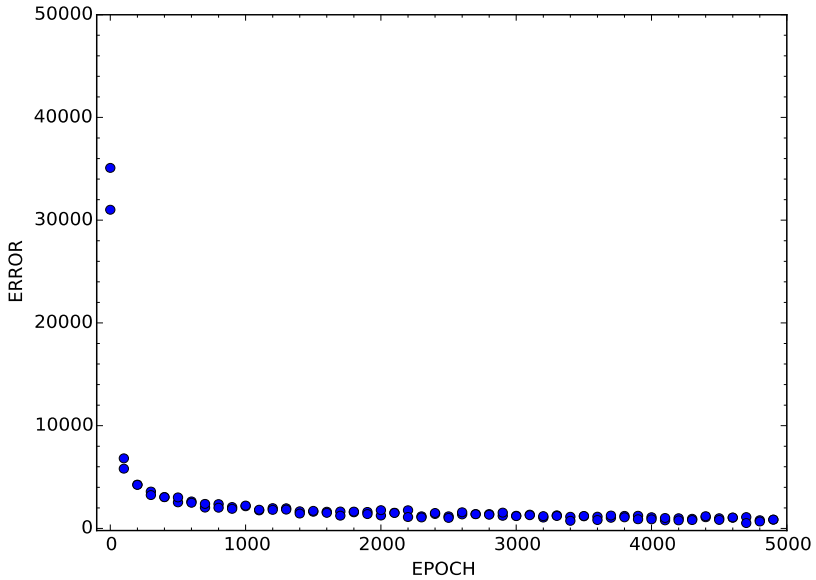


Figura 36. Error entrenamiento Codas con 100 neuronas por capa y 5000 epochs

Melodía	Armonía	Originalidad	Forma	Textura
5	4	5	5	4

Tabla 10. Parámetros de evaluación de los resultados de entrenamiento de Codas con 100 neuronas por capa y 5000 epochs

4.2.5. Resultados del Entrenamiento Final para Desarrollos

- Resultado de entrenamiento con 300 neuronas y 5000 epochs:

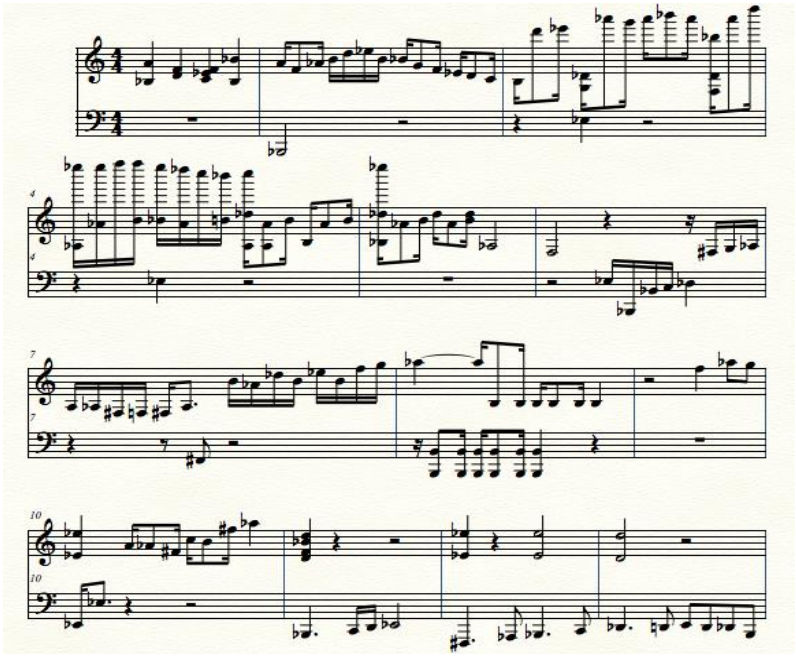


Figura 37. Resultado entrenamiento Desarrollos con 300 neuronas por capa y 5000 epochs

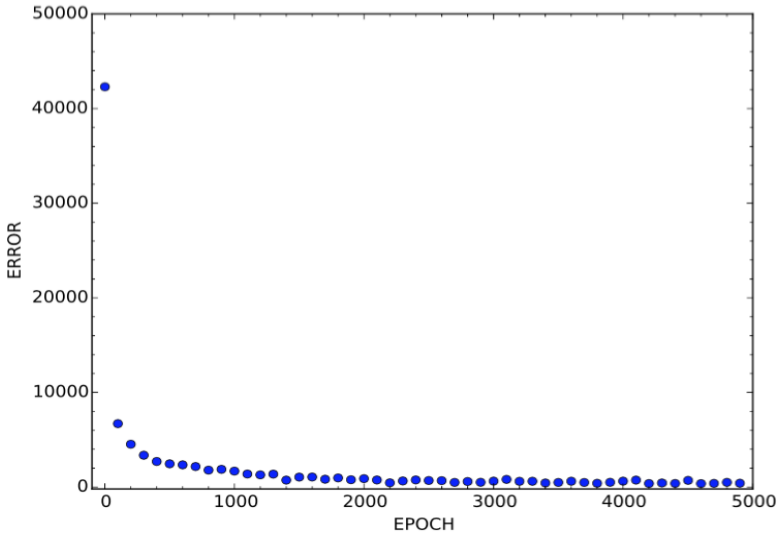


Figura 38. Error entrenamiento Desarrollos con 300 neuronas por capa y 5000 epochs

Melodía	Armonía	Originalidad	Forma	Textura
3	3	5	3	3

Tabla 11. Parámetros de evaluación de los resultados de entrenamiento de Desarrollos con 300 neuronas por capa y 5000 epochs

- Resultado de entrenamiento con 100 neuronas y 5000 epochs:

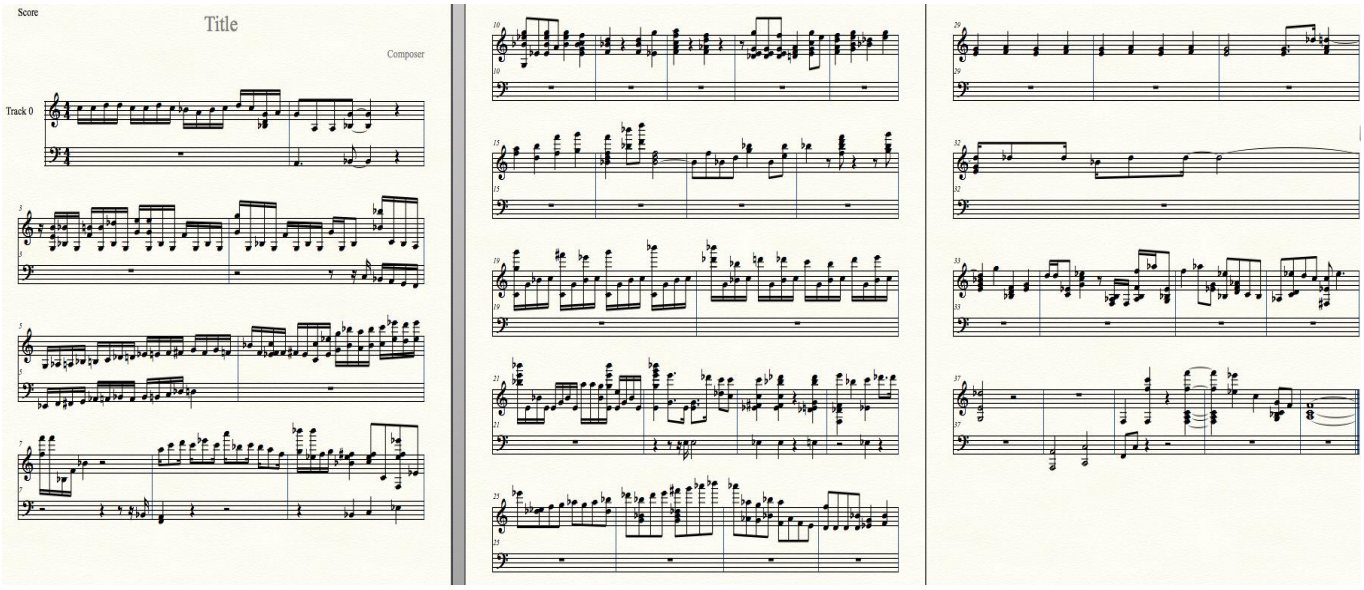


Figura 39. Resultado entrenamiento Desarrollos con 100 neuronas por capa y 5000 epochs

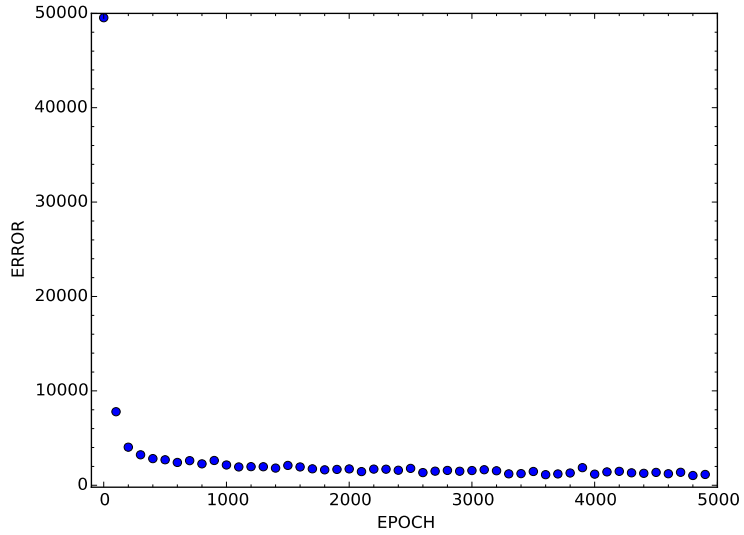


Figura 40. Error entrenamiento Desarrollos con 100 neuronas por capa y 5000 epochs

Melodía	Armonía	Originalidad	Forma	Textura
3	4	5	4	4

Tabla 12. Parámetros de evaluación de los resultados de entrenamiento de Desarrollos con 100 neuronas por capa y 5000 epochs

Como se puede comprobar, el entrenamiento con 100 neuronas da mejores resultados de entrenamiento y a la hora de generar nuevas piezas a partir de los parámetros obtenidos el resultado de los desarrollos no es tan bueno como el del resto de temas.

Los parámetros con los que se consigue un mejor entrenamiento serán **100 neuronas** y **5.000 epochs**. En la siguiente tabla se muestra un resumen de las valoraciones de cada entrenamiento.

	Parámetros	Melodía	Armonía	Originalidad	Forma	Textura
Temas A	300n/capa 2.000 e	1	2	2	1	2
	300n/capa 5.000 e	4	4	5	2	3
	300n/capa 10.000 e	5	5	1	5	5
	100n/capa 5.000 e	5	4	5	5	5
Temas B	300n/capa 5.000 e	3	4	5	4	5
	100n/capa 5.000 e	5	5	5	5	5
Codas	300n/capa 5.000 e	3	3	5	3	4
	100n/capa 5.000 e	5	4	5	5	4
Desarrollo	300n/capa 5.000 e	3	3	4	3	3
	100n/capa 5.000 e	3	4	5	4	4

Tabla 13. Resumen de las valoraciones según los temas y parámetros de entrenamiento

Como resultado final, se ha obtenido una pieza que musicalmente posee fragmentos melódicos y armónicamente buenos, aunque esto no es así a un nivel global de la obra. En cuanto a la estructura final de la obra, se genera a partir de la concatenación de los distintos temas (tema A, tema B, coda y desarrollo) una vez obtenidos, por lo que la red no posee la capacidad de aprender la estructura de la obra, que por otro lado es debido a que solo se entrenan cuatro compases de cada obra debido al coste computacional que requeriría añadir más información a la entrada.

La obra final se ha generado concatenando los archivos resultantes de entrenamiento de Temas A, Temas B, Desarrollos y Codas, por tanto consta de 285 compases con una duración total de 9 minutos 30 segundos y cuya partitura se recoge en el anexo IV.

Capítulo 5

Conclusiones y Líneas Futuras

En este capítulo se realiza una valoración final del trabajo y se exponen ideas de futuro aplicables al Software que permitirían incrementar su usabilidad y expansión.

5.1. Conclusiones

Tras establecerse las bases del entrenamiento de una red neuronal para la composición musical, se ha compuesto una obra resultado del entrenamiento de la red y se ha diseñado una interfaz gráfica de ayuda a la composición, por lo tanto, se han conseguido los objetivos inicialmente planteados en el capítulo 1.

Tras la realización de este trabajo se puede concluir que la red es capaz de componer música aunque sea necesaria una postproducción en la que se concatenen los fragmentos de música obtenidos. Se ha de comentar que la estructura musical estudiada en este trabajo, la Forma Sonata, es la forma musical más compleja que existe, ya que de ella derivan los conciertos para instrumentos como el violín, o las sinfonías que son equivalentes a sonatas para orquesta.

Examinando los resultados, para haber entrenado la red tan sólo 4 compases por obra en una subdivisión de semicorchea (16 notas por compás), los resultados son satisfactorios. Entrenar las obras completas sería muy costoso computacionalmente y aumentando la subdivisión aumentaría este coste, y aunque es cierto que subdividiendo en semicorcheas se pierde información (puede haber pasajes con notas más rápidas), lo normal es que la duración de las notas no exceda la semicorchea.

En resumen, la música generada puede considerarse como válida aunque el hecho de entrenar las obras por partes hace que las transiciones no sean acordes a las piezas generadas (no existe un puente de unión entre la exposición y el desarrollo como suele haber en las sonatas clásicas.

5.2. Líneas Futuras

Tras el análisis de la red y el cálculo de las neuronas y epochs, podrían añadirse a la red nuevas funcionalidades. Como primera mejora podría modificarse el código para que la red se entrenase teniendo en cuenta los canales MIDI, de tal forma que pudieran generarse obras para más de un instrumento. Dentro de este concepto surgen 2 de los trabajos más importantes en la composición, la **instrumentación**, que es la elección de los instrumentos que componen una pieza musical, y la **orquestación**, es decir, la asignación de las voces para cada instrumento.

A su vez, podría incrementarse el número de compositores y la variabilidad de las obras y compases a entrenar y generar, pudiendo entrenar, por ejemplo, valses en compases ternarios, etc, y que el programa fuese capaz de no sólo generar música, sino de decidir qué instrumentos lo componen y qué voces pertenecen a cada instrumento.

En cuanto a la interfaz gráfica, se podría realizar una aplicación móvil, por ejemplo para Apple mediante Swift, de tal manera que los parámetros finales de entrenamiento y la base de datos estuvieran en un servidor para que el propio teléfono pudiese generar música y la aplicación no ocupase mucha memoria.

Por último, esta aplicación podría extrapolarse a otros tipos de música como la música pop o "de consumo", que por su propia estructura sería más sencilla de generar.

6. Bibliografía

[1] POSE, Marcos Gestal. *Introducción a los algoritmos genéticos*. Departamento de Tecnologías de la Información y las Comunicaciones Universidad de Coruña. 2000.

[2] ORTIZ, Juan Antonio Pérez. *Modelos predictivos basados en redes neuronales recurrentes de tiempo discreto*. 2002. Tesis Doctoral.

[3] http://deeplearning.net/tutorial/lstm.html (último acceso Mayo 2017).

[4] https://es.wikipedia.org/wiki/Perceptron (último acceso Mayo 2017).

[5] http://neuralnetworksanddeeplearning.com/chap1.html (último acceso Mayo 2017).

[6] http://www.hexahedria.com/2015/08/03/composing-music-with-recurrent-neural-networks (último acceso Mayo 2017).

[7] ftp://decsai.ugr.es/pub/usuarios/castro/Actividades/Redes-Neuronales/Apuntes/Apuntes%20Javier%20Rodriguez%20Blazquez/Redes%20Multicapa%20-%20Algoritmo%20Backpropagation.pdf (último acceso Mayo 2017).

[8] GOODFELLOW, I., BENJIO, Y., COURVILLE A. *Deep Learning* (Chapter 10). MIT Press. 2016.

[9] http://www.kunstderfuge.com/mozart.htm (último acceso Mayo 2017).

[10] http://www.el-atril.com/midi.htm (último acceso Mayo 2017).

[11] MOOR, James. *The Turing test: the elusive standard of artificial intelligence*. Springer Science & Business Media, 2003.

ANEXO I. MÉTODOS DE COMPOSICIÓN ALGORÍTMICA EN LA INTELIGENCIA ARTIFICIAL

1. Cadenas de Markov

El concepto de Modelo de Markov (MM) fue introducido por Andréi Andreyevich Markov (1856-1922), matemático ruso conocido por sus trabajos en la teoría de las probabilidades, los números y el análisis.

Una cadena de Markov es un modelo matemático que representa la evolución de un evento en el tiempo. En una cadena de Markov el estado es completamente observable.

Para comprender el concepto de modelo de Markov es necesario introducir el concepto de proceso Estocástico.

1.1 Proceso Estocástico

Se define un Proceso Estocástico como aquel comprendido por una sucesión de valores aleatorios X_1, X_2,..., X_n, que no se pueden predecir con exactitud y que dependen del tiempo. X_1 es el estado inicial y X_n es el estado en el instante de tiempo n. Cada instante de tiempo t tiene un valor aleatorio asignado: X.

Cuando el proceso estocástico está compuesto por un número entero de estados se denomina cadena estocástica. Una Cadena de Markov es una cadena estocástica donde la probabilidad del estado siguiente X_{n+1} depende del estado actual X_n. Dicha probabilidad se define como *probabilidad de transición* del estado X_{tn} en un tiempo tn, al estado X_{tn+1}, y es:

$$P(X_{tn+1} = j \mid X_{tn} = i\ ) = p_{ij}(t_n, t_{n+1})$$

La suma de todas las probabilidades de transición en un estado debe ser igual a 1.

Para representar una cadena de Markov existen 2 métodos: los gráficos de transición y las matrices de transición.

Especificación de una cadena de Markov:

- Conjunto de estados $S = \{S_1,..., S_n\}$
- Vector de probabilidades iniciales $\pi = \{ \pi_1,..., \pi_n\}$
- Matriz de transición $A = \{a_{ij}\}$
- Forma compacta $\lambda = \{A, \pi\}$

Los modelos de Markov pueden ser de primer orden, donde la matriz de transición se calcula a partir de un solo evento anterior, o de orden superior, donde más de un evento anterior es utilizado para calcular dicha matriz.

1.2 Cadenas de Markov en la Composición Algorítmica

La primera máquina, Illiac Suit, en componer música a partir de cadenas de Markov fue implementada por Hiller e Isaacson en 1957. Hiller, en colaboración de Baker, desarrolló otro equipo en 1963 denominado Computer Cantata.

A su vez, Iannis Xenakis (1922-2001), compositor e ingeniero civil, compuso a partir de cadenas de Markov, las obras *Analogiques A & B* [1] en 1958 y 1959. Analogique A es una obra acústica para 9 instrumentos de cuerda, y Analoguique B es una obra electroacústica. Xenakis utiliza la altura, densidad y dinámicas como parámetros fundamentales de la obra. Las alturas son notas de la escala en Analogique A y frecuencias en Analogique B. En la tabla 1 se muestran 2 de las 6 alturas utilizadas en Analogique A.

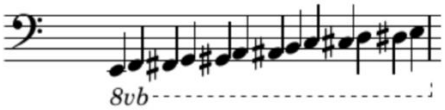
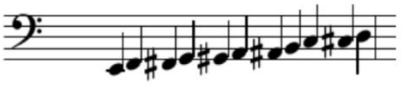
Nombre	Alturas	Grupo
AI		f_0
AII		f_0

Tabla 1. Alturas en *Analogique A* de Iannis Xenakis (1958)

El grupo de cada altura (2 para analogique A) se escoge según unas reglas markovianas y se elige una altura concreta de una categoría de alturas. En cuanto a la densidad, Xenakis opta por medir eventos por unidad de tiempo, de tal forma que obtiene 3 densidades: un evento, 3 eventos y 9 eventos por tiempo, siendo el tiempo medio el compás de 4/4 a ♩= 50. Se consideran como dinámicas ***pp***, ***f*** y ***fff***, y 2 matrices de cambios de estado para alturas (α y β), 2 para dinámicas (γ y ε) y 2 últimas (λ y μ) para densidades. Estas matrices se muestran en la figura 1. Una vez obtenidas todas las matrices mencionadas anteriormente, se obtiene la matriz general de cambio de estado para Analogiques A y B (tabla 2).

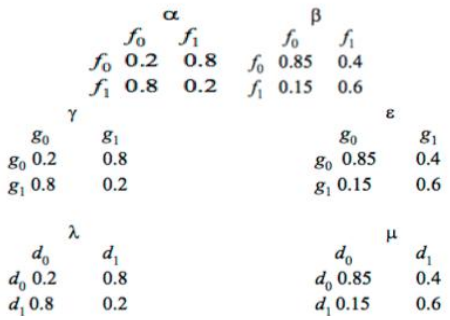


Fig. 1. Matrices de transición α y β, γ y ε, λ y μ para Analogique A © Iannis Xenakis

	A	B	C	D	E	F	G	H
A	0,021	0,357	0,084	0,189	0,165	0,204	0,408	0,096
B	0,084	0,089	0,076	0,126	0,150	0,136	0,072	0,144
C	0,084	0,323	0,021	0,126	0,150	0,036	0,272	0,144
D	0,336	0,081	0,019	0,084	0,135	0,024	0,048	0,216
E	0,019	0,063	0,336	0,171	0,110	0,306	0,102	0,064
F	0,076	0,016	0,304	0,114	0,100	0,204	0,018	0,096
G	0,076	0,057	0,084	0,114	0,100	0,054	0,068	0,096
H	0,304	0,014	0,076	0,076	0,090	0,036	0,012	0,144

Tabla 2. Matriz general de cambio de estado en *Analogiques A y B* de Iannis Xenakis (1958) © Biblioteca General de la Universidad Católica de Argentina

Otro método de componer música con cadenas de Markov puede ser el de generar una melodía a partir de una pieza dada, como muestra Gustavo Díaz-Jerez [2]. En lo que a la melodía concierne, se puede generar una melodía a partir de una dada mediante una cadena de Markov. Cabe destacar que el compás de la melodía y el ritmo de cada nota escogida serán determinados por el compositor al inicio del proceso.

Para generar la melodía se escoge una frase y se enumera cada nota de la frase melódica. A continuación se cuenta la cantidad de veces que se repite la misma nota en la frase, así se puede calcular la probabilidad de cada nota dividiendo el número de veces que se repite por el número total de notas. Se construye una matriz de primer orden contando las veces que cada nota Y precede a una nota X. Como la suma de cada fila de la matriz ha de ser 1, se divide cada elemento por filas por la suma total de cada fila. Una vez elegida la nota inicial, se generará un número aleatorio entre 0 y 1 con el que se elegirá la nota siguiente, repitiéndose el mismo proceso hasta la última nota.

Se puede generar una melodía a partir de varias melodías repitiendo el mismo proceso.

Cuanto mayor es el orden de la cadena de Markov, mejor elaborada estará la melodía resultante, así como los puentes[1] entre 2 frases y la cadencia final[2].

1.3 Modelos Ocultos de Markov (HMM)

Cuando el estado en un modelo de Markov es parcialmente observable, y de orden 1, se denomina Modelo Oculto de Markov.

Especificación de un Modelo Oculto de Markov:

[1] Denominación en análisis musical a la unión entre 2 frases.

[2] Sucesión de acordes que termina la última frase de la obra. En la mayoría de los casos resuelve en la fundamental o primer grado.

- Conjunto de estados $S = \{S_1,..., S_n\}$

- Vector de probabilidades iniciales $\pi = \{\pi_1,..., \pi_n\}$

- Matriz de transición $A = \{a_{ij}\}$

- Vector de probabilidades de salida $B = \{b_{ij}\}$

- Forma compacta $\lambda = \{A, B, \pi\}$

Los HMM presentan 3 problemas: el cálculo de la probabilidad de observar la secuencia dado el modelo, elegir la secuencia de estados que mejor explica las observaciones y el ajuste de parámetros del modelo para maximizar la probabilidad de observar el conjunto de entrenamiento dado el modelo.

El primer problema puede resolverse mediante una resolución directa o mediante un algoritmo. Para el segundo problema existen varios criterios, como el de elegir en cada instante de tiempo el estado más probable, y se resuelven a partir del algoritmo de Viterbi. El tercer problema utiliza el algoritmo de Baum-Welch para su resolución.

En la composición automática, los HMM han sido utilizados para generar contrapunto en relación al *cantus firmus*, de la mano de Mary Farbood y Bernd Schoner [3], e improvisaciones de jazz, de Martin Hirzel y Daniela Soukup.

Para la generación de contrapunto, se crea un HMM en el que la secuencia observable se obtiene del cantus firmus, y el algoritmo de Viterbi calcula los estados ocultos, unos valores de nota que forman el nuevo contrapunto.

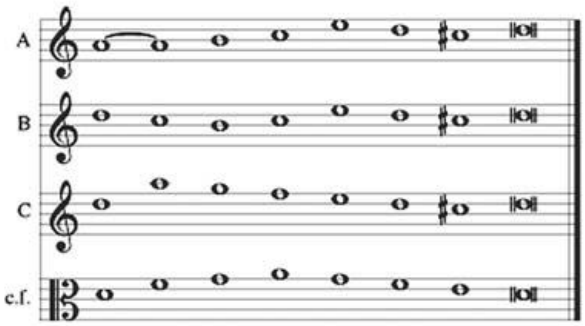


Fig. 2. Ejemplos de contrapunto para un cantus firmus dado © M. Farbood y B. Schoner
A. Compuesto por David Lewin
B y C. Compuestos por la máquina

Existe también el denominado *modelo jerárquico*, a través del cual se puede armonizar una melodía. Moray Allan [4] utiliza modelos de Markov y HMM en la armonización de líneas melódicas, como la voz soprano en un coral de Bach. En este caso, la secuencia observable sería la voz soprano que se quiere armonizar y el algoritmo de Viterbi resolvería la estructura armónica.

Para llevar a cabo este proceso son necesarios 3 HMM. Primero, un HMM construye una estructura armónica (figura 3). La voz soprano es la secuencia observable y la estructura armónica, el estado oculto. Un segundo HMM toma como secuencia observable la estructura armónica y genera acordes, estados ocultos. La ornamentación es realizada por un tercer HMM, que se encarga de asignar símbolos armónicos y de nota del estado actual y siguiente. En la figura 4 se puede observar la armonización de la melodía "Dank sei Gott in der Höhe", de J.S.Bach.

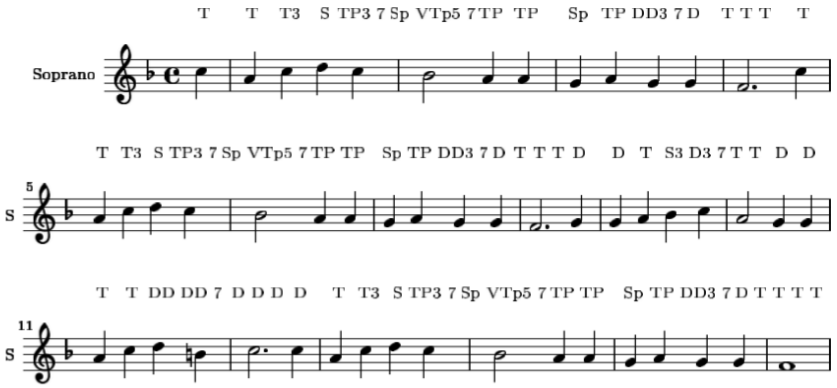


Fig. 3. Estructura armónica generada por el primer HMM © Moray Allan

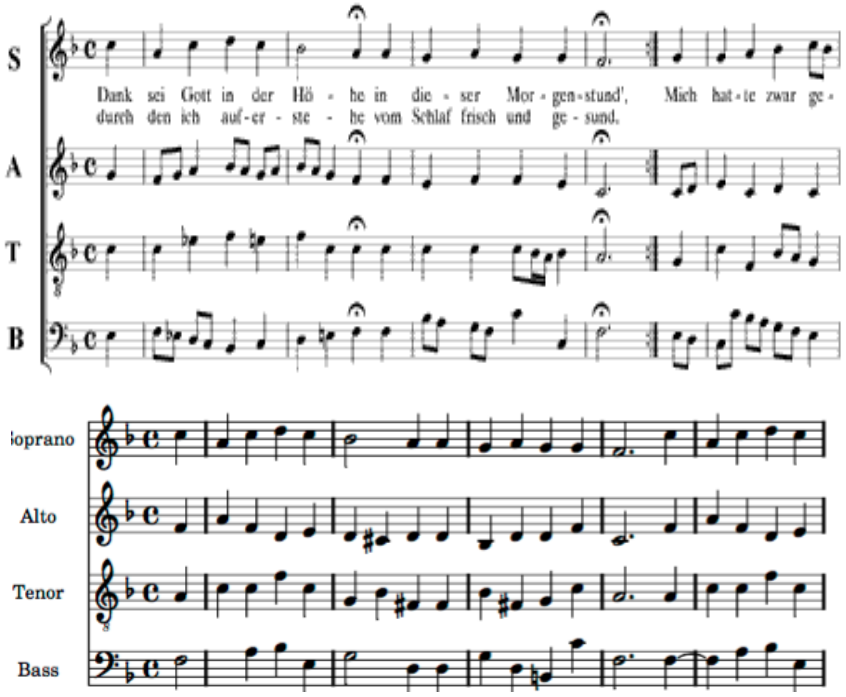


Fig. 4. BWV 288 Bach Original y armonizado con HMMs © Moray Allan

2. Gramáticas Generativas

Noam Abraham Chomsky (1928, -), lingüista y filósofo estadounidense, es el fundador de la lingüística generativa y transformacional. En 1957 publicó la obra *Estructuras Sintácticas*, con la que postuló la existencia de una Gramática Universal.

Chomsky postula con esta teoría, la existencia de una estructura mental innata en los seres humanos, que permite la producción y comprensión de cualquier manifestación lingüística en cualquier idioma. La gramática generativa trata de averiguar el conocimiento que posee el hablante de su lengua, el cual le permite construir y comprender oraciones que nunca ha oído ni leído.

Chomsky afirma que no es suficiente comprender todas las palabras que componen una oración, sino que también es necesario comprender su información sintáctica para comprender el significado total de la misma.

2.1 Jerarquía de Chomsky

Chomsky distingue 4 tipos de gramáticas generativas según sus restricciones. Cada grupo incluye las gramáticas del grupo siguiente:

$$G3 \subset G2 \subset G1 \subset G0$$

Los tipos de gramáticas sirven para generar lenguajes formales. Cada tipo de gramática corresponde a un nivel de capacidad generativa, y cuanto más alta sea esta capacidad, más variedad de expresiones podrán generarse.

Un lenguaje puede generarse por muchas gramáticas diferentes, pero cada gramática describe un lenguaje único.

Una gramática generativa está definida por la cuádrupla (Vt, Vn, S, F), donde Vt es un alfabeto de símbolos, Vn de variables, S un símbolo perteneciente a Vn y F un conjunto de reglas. Una de las formas de definir la jerarquía de una gramática generativa es (|X| es el tamaño del vector X):

Tipo 0 (phrase-structure): unrestricted

Tipo 1 (length-increasing or context-sensitive):

$|P| \leq |Q|$ except possibly for the rule S —> λ

Tipo 2 (context-free): $|P| = 1$ y $|P| \leq |Q|$ except possibly for the rule S —> λ

Tipo 3 (regular or finite-state): every rule has form either

X —> a Y

o Z —> b, donde X, Y, y Z son variables y *a* y *b* terminal symbols.

2.1.1 Gramáticas tipo 0

Este tipo de gramáticas se conocen como *lenguajes sin restricciones*. La capacidad generativa es muy alta respecto a los demás tipos de gramáticas, y su complejidad, infinita. Un ejemplo de este tipo de gramáticas son las redes de Petri.

2.1.2 Gramáticas tipo 1

Los lenguajes generados por este tipo de gramáticas se conocen como *dependientes del tiempo.* La capacidad generativa es muy alta respecto a los demás tipos de gramáticas, y su complejidad, exponencial.

2.1.3 Gramáticas tipo 2

La mayor parte de los lenguajes de programación pueden describirse mediante este tipo de gramáticas. La capacidad generativa es media respecto a los demás tipos de gramáticas, y su complejidad, polinómica.

2.1.4 Gramáticas tipo 3

Este tipo de gramáticas se conocen como *lenguajes regulares.* La capacidad generativa es baja respecto a los demás tipos de gramáticas, y su complejidad, lineal.

Estas gramáticas se clasifican en: Gramáticas lineales por la derecha y Gramáticas lineales por la izquierda.

Las cadenas de Markov mencionadas posteriormente, son un ejemplo de este tipo de gramáticas.

2.2 Gramáticas Generativas en la Composición Automática

Las gramáticas generativas pueden emplearse en la música occidental, en el análisis y composición musical. En jazz, por ejemplo, se crean progresiones de acordes basadas en las reglas de la armonía del jazz y su estructura.

Una estructura musical formada a partir de una gramática generativa estará siempre bien estructurada, pero no siempre cumplirá con la estructura que musicalmente sea correcta. Por ejemplo, el cambio de un acorde perfecto mayor a uno menor puede tomarse como regla, de tal forma que la gramática esté donde esté ese cambio lo considerará correcto, pero normalmente, en armonía, será correcto sólo si está al final de la obra.

Otro de los problemas de las gramáticas generativas, son reglas musicales o armónicas que no pueden ser reproducidas por dichas gramáticas, por ejemplo, en un pasaje con una clave diferente a la del resto de la obra, la gramática generativa haría caso omiso a esta representación produciendo acordes no válidos para la tonalidad dada.

2.2.1 Análisis

Heinrich Schenker, uno de los predecesores de las gramáticas generativas en la aplicación musical, propone que los elementos de una estructura tonal estén referidos a un patrón denominado "Ursatz". El ursatz está formado por 2 voces que descienden progresivamente desde una nota inicial, formando la estructura musical.

A partir de este método, Stephen Smoliar desarrolló un sistema de representación de estructuras musicales complejas mediante árboles.

Otros de los autores que han hecho aproximaciones analíticas de las funciones armónicas han sido Lerdahl y Jackendoff. Así, definen la "estructura de agrupación", que divide la pieza en motivos, frases y secciones, la "estructura métrica", que controla la frecuencia de tonos fuertes y débiles, la "reducción espacio-tiempo", que asocia a cada tono una estructura métrica y de agrupación, la "reducción de prolongación". Un ejemplo de la reducción espacio-tiempo se muestra en la figura 5.

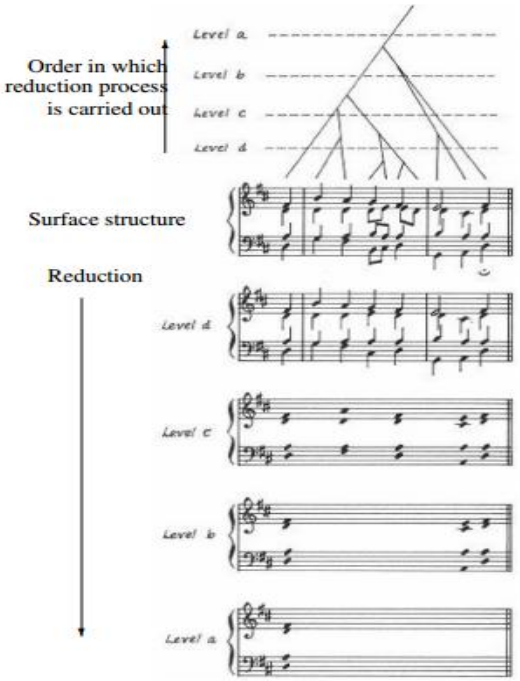


Fig. 5. Reducción del coral "O Haupt voll Blut und Wunden" de la Pasión de San Mateo de J. S. Bach [5, p. 254]

2.2.2 Música Occidental

Gary Rader, usó las gramáticas generativas en el modelado de estructuras armónicas y melódicas, creando rondas. Los grados I, II, III, IV, V y VI son la base de la estructura armónica, y se enlazan entre ellos siguiendo una serie de probabilidades A% a Z%, como en la figura 6.

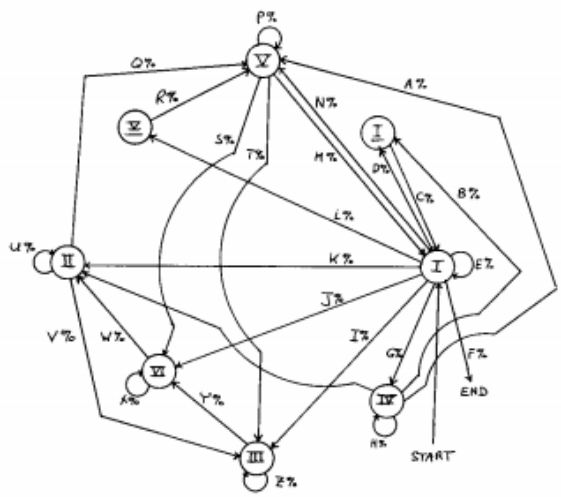


Fig. 6. Progresión armónica según Gary Rader © Gary Rader [6]

Otros autores como Johan Sundberg y Björn Lindblom usaron las gramáticas generativas en la composición de piezas musicales para niños y folk. Como primer experimento tomaron 8 compases de canciones del siglo XIX y las analizaron para generar una gramática para la producción musical.

2.2.3 Procesador de Bol

El comienzo del procesador de Bol [7] comenzó en la India con el estudio de un conjunto de bateristas. Bernard Bel y Jim Kippen, diseñaron el procesador para la notación e improvisación de patrones rítmicos con el esquema *qa'ida* de la tabla (instrumento de percusión), y evolucionó en un sistema capaz de producir música a partir de gramáticas generativas. Dicho sistema fue implementado en el Apple™ IIc.

Muchas piezas de la música de la tabla se transcriben con sílabas onomatopéyicas, en representación de los sonidos y golpes de los tambores, llamados *bols*. Un alfabeto utilizado en composiciones para tabla es de la forma:

$$Vt = \{tr, kt, dhee, tee, dha, ta, ti, ge, ke, na, ra, -\},$$

donde Vt es un conjunto no vacío de símbolos, y el símolo – corresponde a los silencios.

Las duraciones de las sílabas son idénticas. Las sílabas son agrupadas en compases. En la figura 7 se muestran las primeras líneas de las variaciones de *qa'ida*, donde se pueden observar 4 compases (columnas), con una densidad de golpeo de 10 golpes por compás (filas).

dha tr kt dha	tr kt dha ge	dha ti dha ge	dhee na ge na
dha tr kt dha	tr kt dha dha	dha ti dha ge	dhee na ge na
dha ti dha tr	kt dha tr kt	dha ti dha ge	dhee na ge na
dha tr kt dha	ti - dha ti	dha ti dha ge	dhee na ge na
dha tr kt dha	ti dha tr kt	dha ti dha ge	dhee na ge na
ti - dha ti	dha dha tr kt	dha ti dha ge	dhee na ge na
ti dha tr kt	dha dha tr kt	dha ti dha ge	dhee na ge na
tr kt dha ti	dha dha tr kt	dha ti dha ge	dhee na ge na
tr kt tr kt	dha dha tr kt	dha ti dha ge	dhee na ge na
tr kt dha tr	kt dha ge na	dha ti dha ge	dhee na ge na

Fig. 7. Primeras líneas de las variaciones de *qa'ida* [7, p. 4]

Este conjunto de sílabas puede estar definido por una gramática de tipo 3, y puede ser representado mediante un *aceptor finito*. El aceptor finito de la figura 7, está representado en la figura 8.

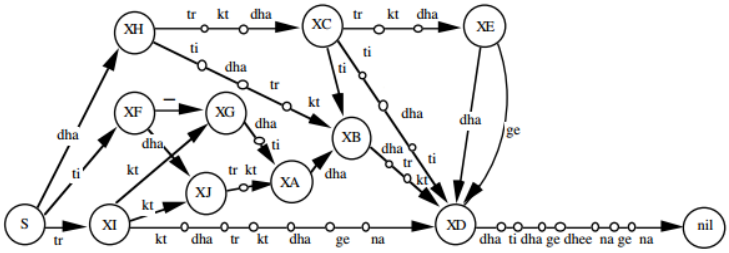


Fig. 8. Aceptor finito de las primeras líneas de las variaciones de *qa'ida* [7, p. 5]

Este aceptor sirve para identificar si una cadena pertenece al conjunto de ejemplos originales, y puede ser utilizado para la generación de nuevas cadenas basadas en el ejemplo. En el aceptor, sólo los estados convergentes o divergentes están representados por nodos de mayor tamaño, siendo los de menor tamaño, otros estados de menor importancia. Esto sugiere una representación alternativa de aceptores de 2 capas, figura 9, para representar la gramática generativa de la figura 10.

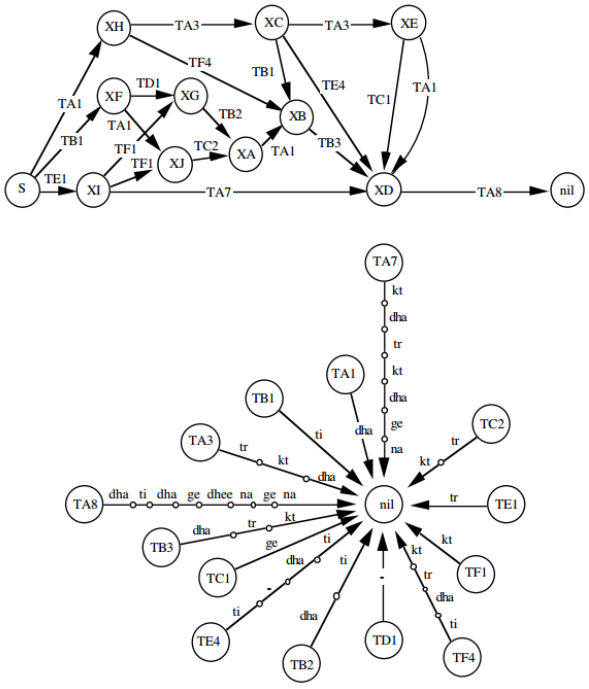


Fig. 9. Aceptor de 2 capas [7, p. 6]

S	—>	TE1 XI			
XI	—>	TA7 XD			
XD	—>	TA8			
XI	—>	TF1 XJ	TA7	—>	kt dha tr kt dha ge na
XJ	—>	TC2 XA	TC2	—>	tr kt
XA	—>	TA1 XB	TE1	—>	tr
XB	—>	TB3 XD	TF1	—>	kt
XI	—>	TF1 XG	TF4	—>	ti dha tr kt
XG	—>	TB2 XA	TD1	—>	-
S	—>	TA1 XH	TB2	—>	dha ti
XH	—>	TF4 XB	TE4	—>	ti - dha ti
XH	—>	TA3 XC	TC1	—>	ge
XC	—>	TE4 XD	TB3	—>	dha tr kt
XC	—>	TA3 XE	TA8	—>	dha ti dha ge dhee na ge na
XE	—>	TA1 XD	TA3	—>	tr kt dha
XE	—>	TC1 XD	TB1	—>	ti
XC	—>	TB1 XB	TA1	—>	dha
S	—>	TB1 XF			
XF	—>	TA1 XJ			
XF	—>	TD1 XG			

Fig. 10. Gramática Generativa equivalente a un aceptor de 2 capas

2.2.4 Jazz

Mark Steedman [8] describió una gramática generativa para la generación armónica de jazz. La salida del sistema consiste en 12 compases de 4/4 generados a partir de reglas de una frase inicial de blues (regla 0). La regla 0 es de la forma S12: I, I7, IV, I, V7, I, y está representada en la figura 11.

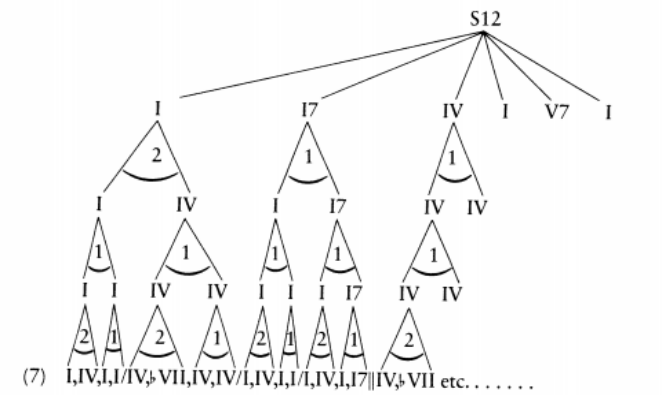


Fig. 11. Secuencia de la regla 0 [8, p. 62]

Existen diferentes tipos de reglas para generar secuencias distintas.

Philip N. Johnson-Laird aplicó las gramáticas generativas en la improvisación de jazz, estableciendo estructuras rítmicas como base, progresiones de acordes y bajos. La línea del bajo está generada a partir de una gramática que produce una melodía, donde el primer, tercer y séptimo grado son los más buscados. El volumen, articulación y timbre no son utilizados para la generación de esta línea.

Otros autores como François Pachet utilizan diferentes aproximaciones para establecer una estructura musical determinada.

3. Autómatas Celulares

3.1 Definición

El concepto de autómata celular (AC) surgió en la década de los años 40 de la mano de John von Neumann. A partir de este concepto, Konrad Zuse y Stanislaw Ulman crearon primer autómata celular.

Un autómata celular [9] es un modelo matemático para un sistema dinámico, compuesto por un conjunto de celdas o células que adquieren distintos estados o valores. Los estados son alterados en tiempo discreto, de tal forma que la *regla de transición local* es definida como la evolución de células a través de una expresión matemática.

Un autómata celular está constituido por un conjunto de estados κ, una función de transición φ, un radio de vecindad r y la configuración inicial c. Así, los autómatas celulares se representan como la cuádrupla:

$$\{\kappa, \varphi, r, c\}$$

Los autómatas celulares poseen 5 elementos fundamentales:

1. **Espacio Regular**. Es el espacio de evoluciones n-dimensional donde cada división es una célula.
2. **Conjunto de Estados**. Se expresa en valores y es aquí donde a cada célula se le asigna un valor.
3. **Configuración Inicial**. Asignación inicial de un estado a cada célula del espacio de evolución inicial.
4. **Vecindades**. Son el conjunto de células y su posición relativa respecto de las demás. Cada vecindad corresponde a un elemento del conjunto de estados.
5. **Función de Transición Local**. Regla de evolución compuesta de una célula central y sus vecindades, que determina el comportamiento del AC.

A su vez, los AC están determinados por fronteras o límites donde son desarrollados. Dichas fronteras o límites son: **Frontera Abierta**, donde las células fuera del espacio del autómata toman un valor fijo, **Frontera Reflectora**, donde estos valores son los que están definidos en el espacio del autómata, **Frontera Periódica**, donde se produce una interacción entre células fronterizas y vecinos, y **Sin Frontera**.

3.2 Tipos de Autómatas Celulares

3.2.1 Autómatas Celulares en Una Dimensión

Los autómatas en una dimensión [10] pueden representarse como (κ, r), donde κ es el número de elementos del conjunto y r el número de vecinos de una célula determinada.

Wolfram define cuatro clases de AC atendiendo l comportamiento de las células a través del tiempo en un diagrama de evoluciones:

- **Clase 1**. Los patrones iniciales evolucionan rápidamente en un estado estable y homogéneo. Las aleatoriedades desaparecen.
- **Clase 2**. Los patrones iniciales evolucionan rápidamente a estados estables u oscilantes. La aleatoriedad no desaparece por completo.
- **Clase 3**. Los patrones iniciales evolucionan pseudo-aleatoriamente.
- **Clase 4**. Los patrones iniciales evolucionan en estructuras que interactúan de forma compleja.

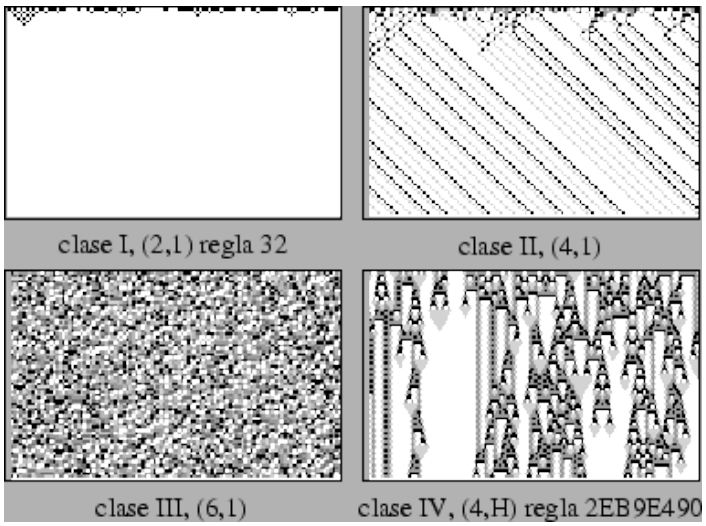


Fig. 12. Clases 1, 2, 3 y 4 de AC de Una Dimensión © 2002 Stephen Wolfram, LLC.

Otro concepto a tener en cuenta en este tipo de AC es la regla 30, denominada así por Stephen Wolfram en 1983 debido a que es el código de Wolfram más pequeño que describe su conjunto de reglas. Esta regla genera patrones aleatorios a partir de reglas simples y bien definidas. En la figura 13 se muestra un ejemplo de regla 30 para un AC de una dimensión.

Célula central y vecindades	Estado al que evoluciona	Representación binaria del conjunto
0 0 0	0	1
0 0 1	1	2
0 1 0	1	4
0 1 1	1	8
1 0 0	1	16
1 0 1	0	32
1 1 0	0	64
1 1 1	0	128

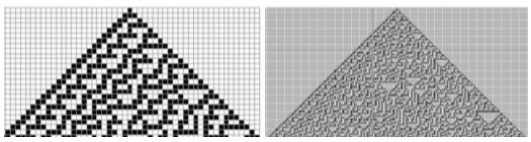


Fig. 13. AC de regla 30 en 25 y 100 generaciones

3.2.2 Autómatas Celulares en Dos Dimensiones

Los AC en dos dimensiones evolucionan en el plano cartesiano. En estos autómatas la función de transición φ está determinada por la vecindad de von Newmann y la vecindad de Moore (figura 14). El espacio de evoluciones se muestra en la figura 14. El ejemplo más común de este tipo de autómatas es ¨El Juego de la Vida¨ de Conway.

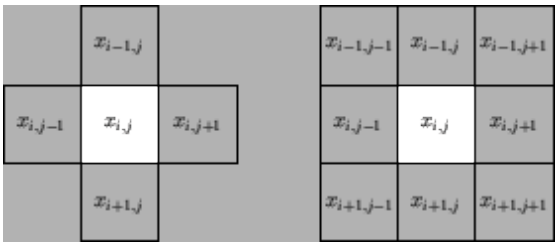


Fig. 14. Vecindad de von Neumann y de Moore

En dicho juego se siguen tres reglas fundamentales:

- **Nacimiento:** Si una célula está muerta o vacía, se reemplaza por una viva si dicha célula tiene 3 vecinos vivos.
- **Supervivencia:** Una célula permanecerá viva si tiene 2 o 3 vecinos vivos.
- **Muerte:** Se reemplaza una célula viva por una muerta si dicha célula no tiene más de un vecino vivo o tiene más de 3 vecinos vivos.

Se puede representar el espacio de evoluciones como se muestra en la figura 15. Las posiciones son ocupadas por elementos del conjunto {0, 1}, donde los huecos en blanco representan el estado 0 y los negros, el 1.

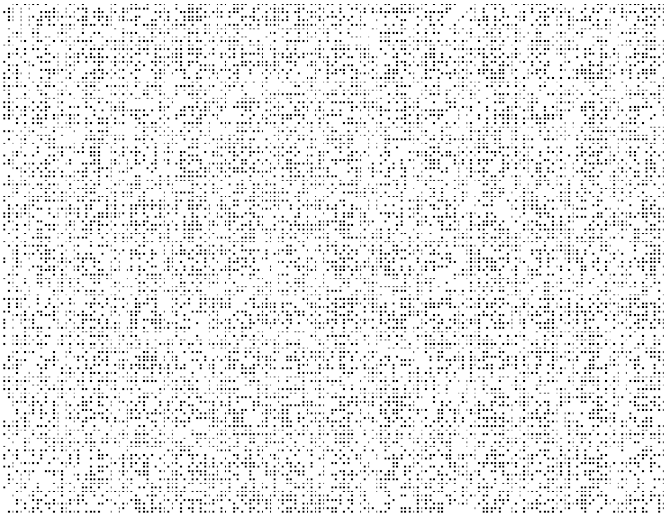


Fig. 15. Espacio de evoluciones en dos dimensiones

3.2.3 Autómatas Celulares en Tres Dimensiones

Este tipo de autómatas, a diferencia del anterior, sólo utiliza la vecindad de Moore (figura 16). Así, estos autómatas tienen 2^{27} vecindades, o $(2^2)^{27}$ reglas de evolución.

Fig. 16. Vecindad de Moore en 3D

3.3 Autómatas Celulares en la Composición Automática

Uno de los primeros autores en utilizar los AC como medio para la composición automática fue Peter Beyls. En uno de sus proyectos, Beyls utiliza más estados iniciales para definir las vecindades y calcular el estado siguiente. Los tonos pertenecientes a una escala determinada, elegida por el usuario, son asignados a las células para realizar el mapeo. Otro proyecto de Beyls, tomando una AC en dos dimensiones, habilita al usuario a cambiar algunos parámetros en tiempo real.

A principios de los 90, Dale Millen desarrolló Cellular Automata Music, utilizando AC en una, dos y tres dimensiones. En otro de sus trabajos explica el comportamiento de AC en una dimensión con 2 estados por célula y una vecindad de 5 células de valores 0 o 1, denominada *K2R2*.

Otra aplicación basada en AC para la composición de música algorítmica es Cellular Automata Workstation, desarrollada por Andy Hunt, Kirk Ross y Richard Orton. En este sistema, se trabaja con un AC en una dimensión con 2 estados para la generación de la estructura musical. El usuario decide el valor de la regla y selecciona un número de posición de las células cuyo desarrollo cronológico se transforma en música. El proceso comienza con la generación de los estados cuyo ritmo es determinado por el usuario y, a continuación, la posición de las células es asignada a valores de tonos MIDI.

CAMUS [11], desarrollado en 1993 por Edardo Reck Miranda, genera estructuras musicales basadas en un autómata en dos dimensiones. Este software aplica la salida del autómata de David Griffeath para resolver El Juego de la Vida de Conway y Demon Cylic Space (DCS).

Para comenzar, se inician tanto el autómata del Juego de la Vida como el DCS. El estado correspondiente al DCS determina la orquestación, es decir, qué instrumento se asigna a cada célula, por ejemplo, el estado 4 estará asignado al instrumento del canal MIDI nº4. La posición de cada célula viva es analizada y utilizada para determinar la tríada en cada tiempo de la composición. Una vez determinada la tríada para cada célula, los estados del Juego de la Vida son utilizados para calcular la posición temporal y la duración de cada nota, construyendo un conjunto de valores a partir de las vecindades de las células, de tal forma que el valor 1 se asigna si la célula está viva y 0 si está muerta:

a = cell (i, j - 1) **b = cell (i, j + 1)** **c = cell (i + 1, j)** **d = cell (i - 1, j)**
m = cell (i - 1, j - 1) **n = cell (i+1, j + 1)** **o = cell (i + 1, j - 1)** **p = cell (i - 1, j + 1)**

A partir de aquí se forman 4 grupos: abcd, dcba, mnop y ponm. Se lleva a cabo la operación OR para la formación de 2 palabras de 4 bits, Tgg y Dur:

Tgg (trigger) = abcd | dcba
Dur (duration) = mnop | ponm

Se utiliza el código AND formando expresiones que representan cada nota de una tríada, siendo **a** la nota inferior, **n** la intermedia y **d** la nota superior:

0000 : a[dn] **0001: [dna]** **0010 : adn** **0011 : dna** **0101 : and**
0110 : dan **0111 : nad** **1001 : d[na]** **1011 : nda** **1111 : n[da]**

Se asocia cada una de las expresiones anteriores a una configuración temporal, como se muestra en la figura 17. Finalmente, se aplica la articulación (hay 22 tipos de articulaciones diferentes), definida por el usuario como parámetro de control donde se especifica la velocidad, dinámica y una secuencia de 12 tonos.

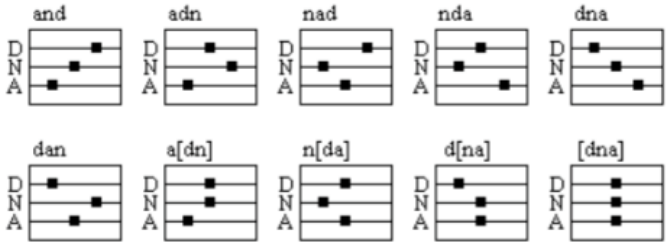


Fig. 17. 10 tipos de códigos temporales

Un ejemplo de obra compuesta por este método es *Entre l'Absurde et le Mystère*.

4. Algoritmos Genéticos

Los algoritmos evolutivos son métodos de optimización y búsqueda de soluciones basados en mecanismos de la evolución biológica. Podemos clasificar los algoritmos evolutivos en estrategias evolutivas (Rechenberg y Schwefel), que utilizan la recombinación, mutación y operación de selección, la programación evolutiva (Fogel), similar a las estrategias evolutivas pero sin utilizar la recombinación y algoritmos genéticos.

Los principios básicos de los Algoritmos Genéticos fueron establecidos por John H. Holland y David E. Goldberg a partir de la teoría de la evolución de Darwin.

4.1 Conceptos Preliminares

Los algoritmos genéticos deben su nombre a los genes, que son unidades de almacenamiento de información genética contenida en el ADN que contienen la información de las funciones de las células del organismo. Los genes son conjuntos de nucleótidos. Los cromosomas son conjuntos de genes, y los genomas, el conjunto de todos los genes.

El genoma humano se compone de 23 pares de cromosomas. La meiosis, conocida como una de las formas de reproducción celular, es la responsable de la variación en el ADN entre los seres vivos.

La teoría de la evolución de Darwin [12], donde se explica que las poblaciones evolucionan durante generaciones mediante la selección natural, es la base de los algoritmos genéticos.

4.2 Definición

John Koza [13] define algoritmo genético como:

"Algoritmo matemático altamente paralelo que transforma un conjunto de objetos matemáticos individuales con respecto al tiempo usando operaciones modeladas de acuerdo al principio Darwiniano de reproducción y supervivencia del más apto, y tras haberse presentado de forma natural una serie de operaciones genéticas de entre las que destaca la recombinación sexual. Cada uno de estos objetos matemáticos suele ser una cadena de caracteres (letras o números) de longitud fija que se ajusta al modelo de las cadenas de cromosomas, y se les asocia con una cierta función matemática que refleja su aptitud"

Los algoritmos genéticos modelan el proceso de evolución como una sucesión de cambios en los genes. Esta técnica se basa en los mecanismos de selección que utiliza la naturaleza, en la que los individuos que más fácilmente se adaptan a los cambios de un entorno determinado sobreviven.

El proceso seguido por los algoritmos genéticos incluye los siguientes pasos [2]:

1. **Inicialización:** Creación de una población inicial aleatoria de n cromosomas (soluciones aleatorias). Cada bit de información que forman los cromosomas se denominan *alelos* (eventos).
2. **Test:** Cada cromosoma es asignado a un valor. Cuanto mayor es ese valor, mayor será la probabilidad del cromosoma de ser elegido en la Reproducción. Si el valor es suficientemente bueno, el proceso termina en este paso.
3. **Reproducción:** Los cromosomas son seleccionados para ser los que generen la nueva solución.
4. **Generación futura:** Se genera una nueva solución. El proceso se repite hasta el segundo apartado hasta que se genera una solución que satisface el problema inicial.

La mutación, variación producida en el patrimonio genético de una especie, introducen variaciones en la población. Un exceso de mutación puede acabar con el valor de selección, y la no mutación implica la no evolución de la población.

4.3 Algoritmos Genéticos en la Composición Algorítmica

Una de las primeras aplicaciones de composición algorítmica fue desarrollada por Andrew Horner y David Goldberg. En esta aplicación se utiliza la técnica *thematic bridging*, que toma un patrón y lo modifica mediante algunas funciones, y compara el resultado con otro patrón. Cada elemento del patrón es un acorde. Un ejemplo de la modificación del patrón inicial puede ser el de eliminar en un primer paso el último elemento del patrón, después, desordenar el patrón aleatoriamente, eliminar el último elemento del patrón, modificar el primer elemento y, volver a desordenar aleatoriamente los elementos restantes. La salida constaría de un patrón con todos los elementos generados por orden de aplicación de las operaciones mencionadas anteriormente.

George Papadopoulos y Geraint Wiggins [14] utilizan los algoritmos genéticos en la generación de melodías de jazz. El objetivo es generar melodías de duración variable y estructura rítmica basada en unos acordes dados. En esta metodología, los cromosomas representan los grados de la escala relativos al acorde actual, siendo una secuencia de pares (grado, duración). Los silencios se representan por la constante *rest*, en lugar de grado. La progresión de acordes, dada como entrada, es una secuencia de {fundamental del acorde, tipo de acorde mayor o menor, duración}, por ejemplo, (Bb, maj7, minim). Las operaciones genéticas utilizadas son **mutaciones locales**, **mutaciones de copia y operación**, y **mutaciones de copia restringida y operación**.

Para comenzar, se prepara la población representada por pares (*fitness*, genotipo). La preparación consiste en 2 pasos, inicialización y evaluación inicial. Para cada cromosoma de la población se genera una secuencia de *m* genotipos. Los silencios se fijan en una probabilidad de aparición del 12.5%.

La función *fitness*, posteriormente, es utilizada para evaluar cada genotipo generado aleatoriamente. Las características de esta función se resumen en la tabla 3.

Intervalos Largos	Por la aleatoria inicialización de cromosomas, puede haber intervalos grandes
Coincidencia de Patrones	Entre tonos, no en el ritmo
Suspensiones	Notas repetidas 2 compases seguidos
Nota en 1er tiempo de compás	La 1ª nota del compás puede ser un acorde (+10), un silencio (+10), nota de la escala (-10), o no perteneciente a la escala (-20)
Nota de mitad de Compás	Igual que en el caso anterior, en compases de 4/4, la tercera nota. Sus valores son (+5), (+5), (-5), (-20)
Contorno	Comparación entre el contorno del cromosoma y el elegido por el usuario
Velocidad	El algoritmo añade notas en cada par de compases, considerando los tiempos: lento, medio y rápido

Tabla 3. Características de la función *fitness*.

La entrada, parámetros y preferencias se resumen en: método de selección, método de unión, tamaño de la población (40 cromosomas), generación máxima, duración de la melodía, progresión de acordes, intervalo más amplio (sexta mayor), coincidencia de patrones, pesos, notas largas, probabilidades de duración y *hill-climbing*.

Un resultado de melodía generada a partir de mutaciones de copia restringida y operación se muestra en la figura 18.

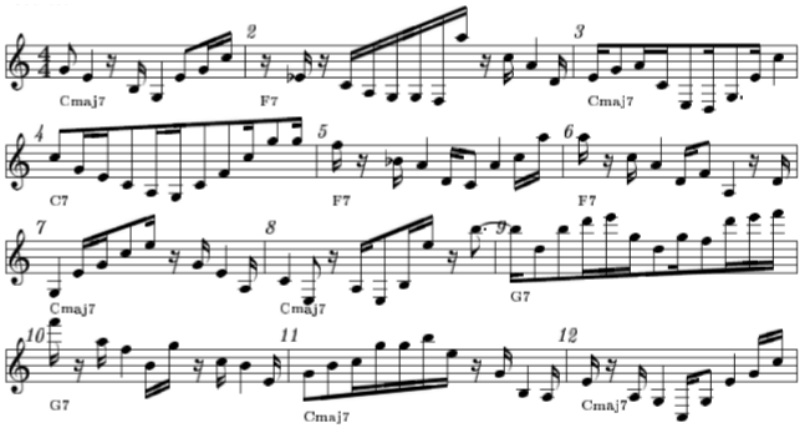


Fig. 18. Melodía generada por el método de Papadopoulos y Wiggins © Papadopoulos y Wiggins.

En otro método para la composición de música mediante algoritmos genéticos, propuesta por Gustavo Díaz-Jerez [2], se comienza tomando un número entero de cromosomas o espacios para eventos, y 4 alelos o eventos: A, B, C y D. Cada alelo representa un tiempo de negra.

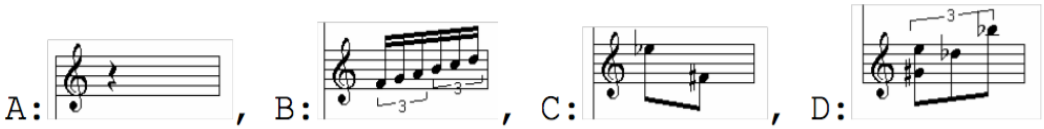


Fig. 19. Alelos iniciales © Gustavo Díaz-Jerez

Siguiendo los pasos descritos en [2], asignando los valores de alelo descritos en la tabla 1, se obtiene una generación 0, generada aleatoriamente. Después, se obtienen más generaciones hasta que el valor asignado en el paso 2, satisface el esquema planteado.

Cromosomas	Probabilidad
Con 4 alelos diferentes	0.3
Que contienen alelos A y B	0.2
Con 4 alelos iguales	0.05
Con 3 alelos iguales	0.08
Con 4 alelos iguales	0.1
Otras posibles combinaciones	0.12

Tabla 4. Valor asignado en el paso 2 a cada cromosoma

Un ejemplo de los resultados obtenidos se muestra en la figura 20.

```
Generation 0 (randomly generated):

BCAB CDBC ACAC ABDC ACCB ADAB ACCB CADB CADB BBDB
```

Generation	Chromosomes
1	ABDB ADAC ACCB BBDB ABDC CDBB CADB BCAC ADAC ABDC
2	BCAC CADB ABDB CADB CDBC CADC ABDC ABDB ABDC ABDC
3	ABDC ABDC ABDC CDBB ABDB CADC ABDC CADB ABDC BCAB
4	ABDC ABDC ABDC ABDB ABDC CADC CADB ABDB CADC CADC
5	CADB CADC ABDC ABDB ABDC ABDB ABDC CADC ABDC ABDC
6	ABDC CADC ABDC ABDC CADC CADB CADC ABDB CADC ABDC
7	CADB ABDB CADC ABDC ABDA ABDC ABDC ABDC CADB ABDC
8	ABDB CADC ABDB ABDC CADC ABDB CADC ABDC ABDB ABDC
9	ABDC ABDB ABDC CADB ADDC ABDB CADC CADC ABDC ABDC

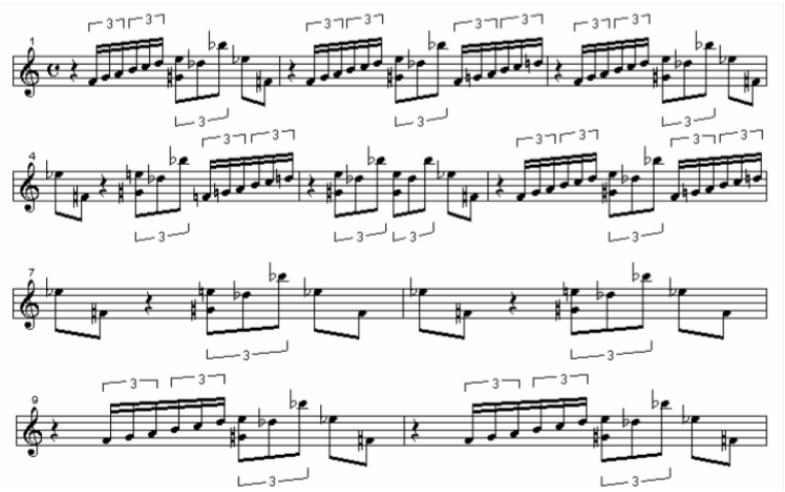


Fig. 20. Resultados de los cromosomas obtenidos y representación musical © Gustavo Díaz-Jerez

También se pueden utilizar algoritmos genéticos para armonizar la voz soprano. S. Phon-Amnuaisuk, Andrew Tuson y G. Wiggins [15] usan los algoritmos genéticos para ello. La representación consiste en una matriz de 5 cadenas de igual tamaño. Las 4 primeras filas son las voces soprano, alto, tenor y bajo, y la última, la duración de los acordes. Esta representación se muestra en la figura 21.

chromosome length

Soprano	[0,0,3]	[0,0,3]	[4,0,2]	[0,0,3]	[1,0,3]	[4,0,2]
Alto	[2,0,2]	[2,0,2]	[2,0,2]	[2,0,2]	[4,0,2]	[1,0,2]
Tenor	[4,0,1]	[4,0,1]	[2,0,1]	[0,0,2]	[7,0,1]	[1,0,1]
Bass	[2,0,1]	[0,0,1]	[0,0,1]	[4,0,1]	[4,0,1]	[7,0,0]
Duration	1	2	1	1	2	2

Fig. 21. Representación de un cromosoma en una obra a 4 voces [11]

Las operaciones de mutación llevadas a cabo para dicha armonización son:

- **Empalme.** Selecciona un punto de cruce entre tonos sucesivos de la melodía y los acordes correspondientes.
- **Perturbación.** Mutar al permitir al alto, tenor y bajo el moverse hacia arribao hacia abajo en un semitono o tono. La selección de las posibles mutaciones es aleatoria.
- **Intercambio.** Mutación mediante el intercambio de 2 voces aleatorias entre alto, tenor o bajo, así es posible el cambio en la inversión de los acordes.
- **Registro.** Mutar a un acorde diferente, generado a partir de la melodía. Los acordes son construidos desde la nota soprano dada como la fundamental del acorde, su 3ª y su 5ª. Cuando el acorde tiene más de 3 notas, se procede a duplicar una de las notas pertenecientes al acorde.
- **Inicio de Frase.** Mutar el inicio de frase para comenzar con la fundamental en la posición más baja del acorde.
- **Fin de Frase.** Mutar el final de frase para terminar en el acorde de la fundamental de la tonalidad.

La función *fitness* se encarga de realizar enlaces entre acordes. Algunos de los enlaces considerados como correctos en este método son aquellos donde se produce un enlace entre acordes por unísono paralelo, quintas perfectas paralelas y octavas paralelas[3], prefiriendo la repetición entre acordes, movimiento de 5ª descendente paralelo, progresión hacia la tónica y retroceso. Dicha función también rechaza aquellos acordes disonantes.

A continuación, en la tabla 5, se muestra una tabla sonde se muestran las probabilidades y enlaces o acordes desechados por el algoritmo.

[3] En armonía, no es correcto utilizar estos enlaces.

Population:	30-200	Migration interval:	20 generations
Operator Probabilities			
P(Splice):	0.3	P(Perturb):	0.1
P(Swap):	0.2	P(Rechord):	0.3
P(PhraseStart):	0.05	P(PhraseEnd):	0.05
Fitness Penalties			
Invalid Pitch:	1	Invalid Chord:	10,1
Invalid Range:	10	Invalid Interval:	10
Invalid Doubling:	10	Voice Crossing:	10
Hidden Unison:	10	Single voice progression:	10
Dual voice progression:	10	Harmonic progression:	10,1
Harmonic Analysis:	100,20		

Tabla 5. Probabilidades y parámetros no válidos en la armonización con algoritmos genéticos.

Un ejemplo de la armonización con algoritmos genéticos es el primer compás del Himno de la Alegría de Beethoven mostrado en la figura 22.

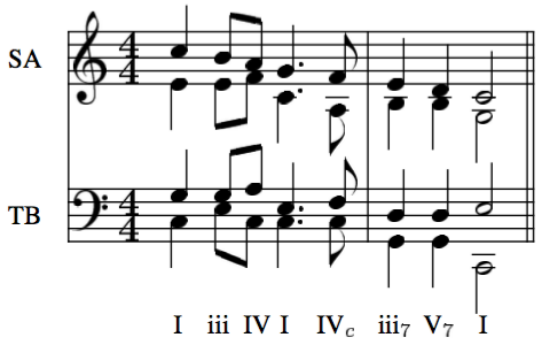


Fig. 22. Armonización de los últimos compases de “Joy to the World”

5. Autosimilitud y Caos

El caos es sinónimo de desorden o falta de estructura. En ciencia, cuando se habla de caos se refiere al *caos determinista*.

La teoría del caos se hizo popular en la década de los años 80 de la mano de Edward N. Lorenz, quien asentó el término "efecto mariposa", y Benoit Mandelbrot, creador de la **Geometría Fractal**.

5.1 Teoría del Caos

La teoría del caos es una rama de la ciencia que estudia sistemas complejos y dinámicos muy sensibles a las variaciones en las condiciones iniciales. El denominado *efecto mariposa*, se resume en que si en un sistema se produce una pequeña perturbación, éste podrá generar un efecto de alta dimensión a corto o medio plazo.

Existe caos cuando sucesos que comienzan con las mismas condiciones iniciales evolucionan de forma diferente, separándose exponencialmente.

Esta teoría de que pequeñas perturbaciones pueden causar grandes efectos fue descrita por Jules Henri Poincaré, quién es conocido además por la conjetura de Poincaré.

A su vez, Pierre-François Verhulst, definió la **ecuación logística**, que es modelo de crecimiento poblacional. Sus soluciones vienen dadas por la expresión:

$$P(t) = \frac{K}{1 + Ae^{-rt}}$$

Para examinar el argumento de que es imposible realizar cálculos de forma exacta matemáticamente, se examina un sistema dinámico muy sencillo, denominado **mapa de Bernouilli**, definido por la expresión:

$$x_{n+1} = (2.x_n)_{mod\ 1}$$

5.2 Atractores Extraños

El *espacio de fases* es el espacio formado por todos los parámetros necesarios para caracterizar el estado de un sistema. El comportamiento de un sistema dinámico puede representarse sobre el espacio de fases, ya que los diagramas de fases no muestran con claridad la trayectoria definida. Se dice que hay un atractor cuando el sistema es "atraído" hacia un tipo de movimiento.

Un atractor en sistemas no caóticos es una forma geométrica. Pueden ser de punto fijo, de ciclo límite o caóticos. En sistemas caóticos, un atractor es un diagrama de soluciones de un sistema caótico, presentando una estructura extraña, de ahí la

denominación de atractor extraño. El atractor extraño más conocido es el atractor de Lorenz (Figura 23).

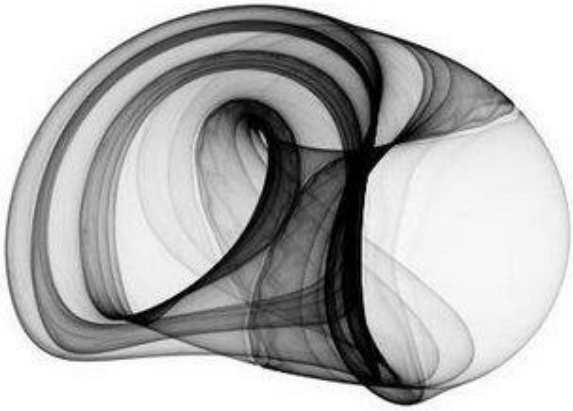


Fig. 23. Atractor de Lorenz en 3D con Chaoscope © Salvador Ruiz Fargueta

5.3 Fractales

Un fractal es un objeto geométrico cuya estructura básica se repite a diferentes escalas. Un objeto geométrico fractal es irregular y autosimilar. Los fractales pueden encontrarse en procesos naturales, como la cristalización.

Georg Cantor en 1883 denominó **conjunto de cantor** al subconjunto fractal del intervalo real [0, 1], que admite las definiciones numérica y geométrica.

Fig 24. Primeras etapas del conjunto de cantor

Otro ejemplo de fractal es la *curva del copo de nieve o estrella de Knoch* (figura 24), denominada así por el matemático sueco Helge von Koch. Es una curva continua cerrada no diferenciable en ningún punto.

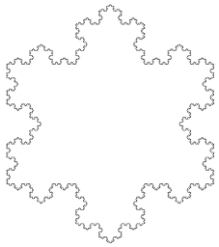


Fig. 25. Curva del copo de nieve o estrella de Knoch

Un **conjunto de Julia**, denominado así por el matemático Gaston julia, es una familia de conjuntos fractales obtenidos al estudiar el comportamiento de los números complejos,

iterados por una función holomorfa. Los conjuntos de Julia son los formados por la familia definida en la ecuación:

$$z(n+1) = z(n)^2 + c$$

Cuando el valor de c es muy pequeño, el conjunto no es computable y se habla del **conjunto de Mandelbrot** (Figura 25), considerado como el prototipo de los fractales.

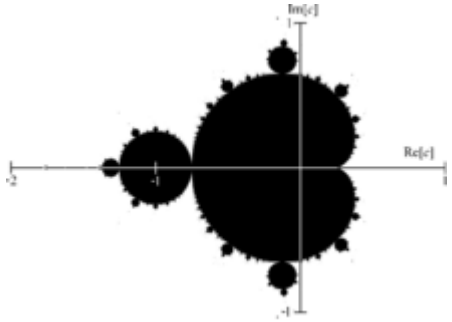


Fig. 26. Conjunto de Mandelbrot

5.4 Sistemas de Lindenmayer

Los sistemas de Lindenmayer (LS) o sistemas-L deben su nombre a Aristid Lindenmayer (1925-1989), quién desarrolló un lenguaje formal que representa el crecimiento de las algas. Paulien Hogeweg y Ben Hesper extendieron este sistema introduciendo su representación gráfica, en 1974.

Como en las gramáticas de Chomsky, los sistemas-L trabajan con reglas de reescritura. Un sistema-L, puede representarse mediante la tripla:

$$\{v, \omega, P\}$$

donde v es el alfabeto, es decir, un conjunto finito de símbolos, ω, el axioma del iniciador, es decir, la cadena que describe al sistema en su situación inicial y P, un conjunto finito de reglas de producción.

La idea básica de funcionamiento del sistema-L, es que al definir un símbolo del alfabeto y su regla de producción, se itera y se obtiene una nueva cadena del lenguaje. Algunas definiciones [16] o **familias del sistema-L** son:

- **Libre de Contexto.** Si la producción se refiere a un solo símbolo.
- **Sensitivo al Contexto.** Si la aplicación de la regla depende de sus vecinos
- Determinista. Si existe una producción para cada símbolo.
- **No Determinista.** Si hay un símbolo para cada producción.
- **Estocástico.** Varias producciones para un mismo símbolo.
- **Con Extensiones.**
- **Temporal.**
- **Propagativo.** Ningún símbolo produce una cadena vacía.
- **No Propagativo.** Algún símbolo produce una cadena vacía.
- **Tabular.**

- **Sensitivos al Ambiente.** En la evolución intervienen factores externos al sistema-L.

5.5 Autosimilitud y Caos en la Composición Algorítmica

Existen diferentes aproximaciones asociadas a la composición de música algorítmica dentro del marco del la teoría del caos, como el **ruido fraccional**. El ruido fraccional describe diferentes tipos de ruido con respecto a sus densidades espectrales ($1/f^n$).

Richard F. Voss y John Clarke describieron características de la densidad espectral en grabaciones de diferentes estilos musicales, mostrando sus paralelismos con el ruido *1/f*. Más tarde, se extendió este modelo a una estructura con 2 voces.

Dodge describió una estructura musical de 3 voces basándose en el ruido *1/f*. Para cada voz, los nuevos tonos son producidos de acuerdo a los diferentes valores de la salida del ruido *1/f*. La segunda línea es creada siguiendo el mismo procedimiento, y así se crea una estructura de densidad creciente.

En cuanto a los **sistemas caóticos**, Jeff Pressing mapeó la trayectoria de una ecuación no lineal en parámetros musicales. La salida la usó para controlar el tono, duración, dinámicas, etc. Rick Bidlack también mapeó, en este caso, sistemas de ecuaciones de 2, 3 y 4 dimensiones en parámetros musicales.

5.6 Sistemas-L en la Composición Algorítmica

P. Prusinkiewicz describió el mapeo de la generación de valores de nota desde una *turtle interpretation* de los sistemas-L, dando un ejemplo a partir de la curva de Hilbert.

John McCormack comparó aproximaciones estocásticas, modelos de Markov, gramáticas generativas y sistemas-L para la composición musical, desarrollando un software estructurado por gramáticas, siendo el tono, duración y timbre diferentes controladores, controlados por una estructura polifónica. El esquema de ester proceso se muestra es la figura 26.

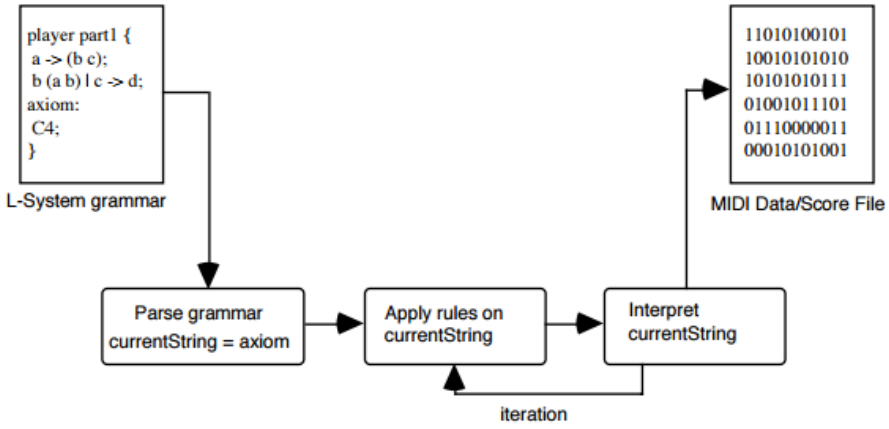


Fig. 27. Diagrama del Sistema de McComarck [17, p. 12]

Roger Luke DuBois describió diferentes posibilidades de mapeo de sistemas-L y desarrolló una aplicación en la que el propio músico suministra la entrada. En este sistema los *turtle graphics* muestran la visualización de la estructura del sistema-L. Un ejemplo de string resultante se muestra en la figura 27, donde ω: X, interación: 5, P1: X -> F-[[X]+cX]+F[+FcX]-X, y P2: F -> FF.

FFFFFFFFFFFFFFF-[[FFFFFFFF-[[FFFF-[[FF-[[F-[[X]+cX]+F[+FcX]-X]+cF-[[X]+cX]+F[+FcX]-X]+FF[+FFcF-
[[X]+cX]+F[+FcX]-X]-F-[[X]+cX]+F[+FcX]-X]+cFF-[[F-[[X]+cX]+F[+FcX]-X]+cF-[[X]+cX]+F[+FcX]-
X]+FF[+FFcF-[[X]+cX]+F[+FcX]-X]-F-[[X]+cX]+F[+FcX]-X]+FFFF[+FFFFcFF-[[F-[[X]+cX]+F[+FcX]-X]+cF-
[[X]+cX]+F[+FcX]-X]+FF[+FFcF-[[X]+cX]+F[+FcX]-X]-F-[[X]+cX]+F[+FcX]-X]-FF-[[F-[[X]+cX]+F[+FcX]-X]+cF-
[[X]+cX]+F[+FcX]-X]+FF[+FFcF-[[X]+cX]+F[+FcX]-X]-F-[[X]+cX]+F[+FcX]-X]+cFFFF-[[FF-[[F-
[[X]+cX]+F[+FcX]-X]+cF-[[X]+cX]+F[+FcX]-X] +FF[+FFcF-[[X]+cX]+F[+FcX]-X]-F-[[X]+cX]+F[+FcX]-X]+cFF-
[[F-[[X]+cX]+F[+FcX]-X]+cF-[[X]+cX]+F[+FcX]-X]+FF[+FFcF-[[X]+cX]+F[+FcX]-X]-F-[[X]+cX]+F[+FcX]-
X]+FFFF[+FFFFcFF-[[F-[[X]+cX]+F[+FcX]-X]+cF-[[X]+cX]+F[+FcX]-X]+FF[+FFcF-[[X]+cX]+F[+FcX]-X]-F-
[[X]+cX]+ F[+FcX]-X]-FF-[[F-[[X]+cX]+F[+FcX]-X]+cF-[[X]+cX]+F[+FcX]-X]+FF[+FFcF-[[X]+cX]+F[+FcX]-X]-F-
[[X]+cX]+F[+FcX]-X]+FFFFFFFF[+FFFFFFFFcFFFF-[[FF-[[F-[[X]+cX]+F[+FcX]-X]+cF-[[X]+cX]+F[+FcX]-

Fig. 27. String resultante de la iteración 5 [18, p. 30] En la figura 28, se muestra el mapeado de tonos de los símbolos Lindenmayer.

Section Letter (Symbol Mapping)									
Pitch Class	A	B	C	D	E	F	G	H	I
C 0	F	[	[	-	(P)	F	c	[	
C#/Db 1		]						]	
D 2	]			+	+	]	[		+
D#/Eb 3	(P)	F		]			(P)	F	
E 4			-		-	(P)	F		-
F 5	+	-	+		[	+		-	[
F#/Gb 6		+					]	+	
G 7	-		(P)	F	c	-	+		c
G#/Ab 8		(P)	F	[					
A 9	c		c	(P)	F	c	-	(P)	F
A#/Bb 10	[	c	]	c	]	[		c	]
B 11									

Fig. 28. Mapeado de tonos en símbolos de Lindenmayer [18, p. 32]

6. Redes de Transición

Las redes de transición (TN) consisten en un conjunto de autómatas y están representadas mediante gráficos, donde los nodos son estados del autómata y las líneas entre nodos, transiciones. Cuando la red posee una estructura que permite procesos recursivos, se denomina **red de transición recursiva**.

Un elemento o marca se propaga a través de la red comenzando en el primer nodo y pasando a través de los nodos adyacentes.

Las **redes de transición aumentada (ANT)** son aquellas en las que la TN se extiende dadas unas reglas o instrucciones. Dichas redes se emplean en la definición de lenguajes formales. Dichas redes incluyen el comando "salto", que permite unir 2 nodos no adyacentes, obviando los nodos entre ellos. La representación de este tipo de redes en el lenguaje se muestra en la figura 29.

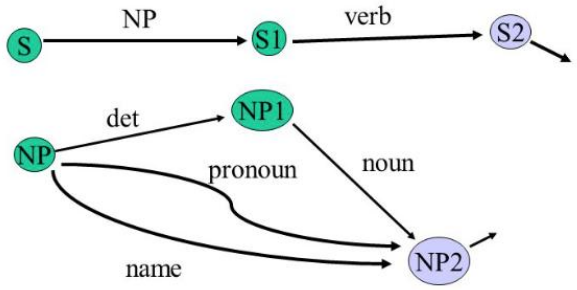


Fig. 29. Representación de una ATN en el lenguaje

6.1. Experiments in Musical Intelligence

Uno de los programas más conocidos en la composición algorítmica es EMI (Experiments in Musical Intelligence), diseñado por David Cope en 1981. EMI es capaz de componer dado un estilo musical. Las obras compuestas por EMI han pasado el *test de Turing,* prueba de habilidad de una máquina para exhibir un comportamiento similar al de un humano.

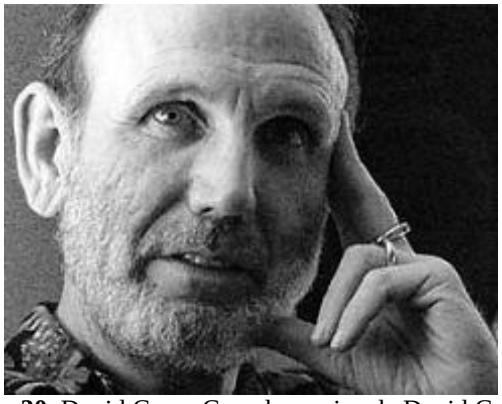


Fig. **30.** David Cope. Con el permiso de David Cope

Al comienzo de la realización del programa, Cope dividía los corales de Bach en segmentos, recombinándolos considerando la voz principal y utilizando transiciones entre segmentos como lo hacía el propio Bach. Antes del análisis, para asegurarse de que se dispone de las entradas suficientes para generaciones nuevas, se representa el coral a una sola clave.

La base de datos de EMI está basada en segmentos o motivos musicales de diferentes estilos. Para el análisis del lenguaje musical se emplea SPEAC, un sistema que otorga flexibilidad y crea enlaces entre acordes y motivos de forma lógica musicalmente. Está basado en los métodos de Heinrich Schenker. Este sistema se compone de:

1. **Estado.** Representan unidades que no son el resultado de ningún proceso en particular.
2. **Preparación.** Gestos que modifican el significado de otros componentes.
3. **Extensión.** Prolongación de estados.
4. **Antecedente.** Prepara una situación musical y requiere soluciones consecuentes.
5. **Consecuente.** Resolución del antecedente.

6.2. Redes de Petri

Las redes de Petri fueron introducidas por Carl Adam Petri en los años 60. Son grafos orientados con dos tipos de nodos, lugares y transiciones unidos mediante flechas. Las redes de Petri son un tipo de TN utilizadas en modelado y análisis de protocolos de comunicación. Un ejemplo de representación de una red de Petri se muestra en la figura 31.

Una red de Petri está definida por la cuádrupla {P, T, A, M}, donde P es el conjunto de lugares, T el conjunto de transiciones, A el conjunto de arcos y M, el marcado de la red.

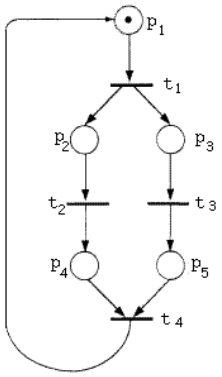


Fig. 31. Representación de una red de Petri

En composición algorítmica, Goffredo Haus y Alberto Sametti desarrollaron **ScoreSynth**, un sistema que procesa información musical con redes de Petri. Este sistema es capaz de manipular y generar datos MIDI de diferentes formas.

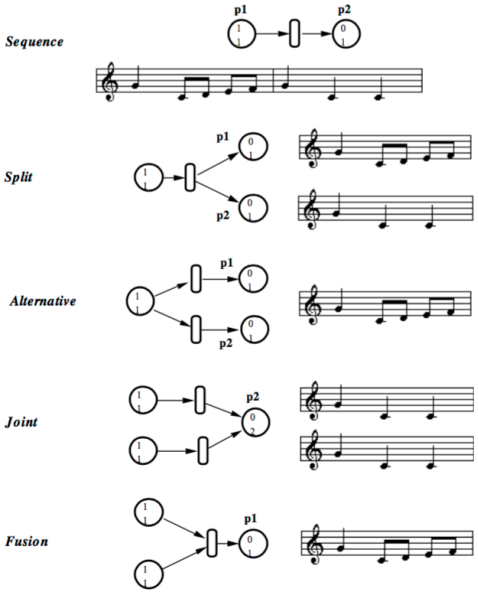


Fig. 32. Estructura musical en redes de Petri

7. Referencias

[1] XENAKIS, Iannis. *Musiques Formelles Nouveaux Principes Formels de Composition Musicale*. 1981.

[2] DIAZ-JEREZ, GUSTAVO. *Algorithmic music: using mathematical models in music composition*. 2000. Tesis Doctoral. The Manhattan School of Music.

[3] FARBOOD, M., SCHONER, B. Analysis and synthesis of Palestrina-style counterpoint using Markov chains. En *Proceedings of the International Computer Music Conference*. 2001. p. 471-474.

[4] ALLAN, Moray. Harmonising chorales in the style of Johann Sebastian Bach.*Master's Thesis, School of Informatics, University of Edinburgh*, 2002.

[5] HIRATA, K., TOJO, S., HAMANAKA, M. An Algebraic Approach to Time-Span Reduction. En *Computational Music Analysis*. Springer International Publishing, 2016. p. 251-270.

[6] RADER, Gary M. *A method for composing simple traditional music by computer*. The MIT Press, Cambridge. 1993.

[7] Bernard Bel, James Kippen. *Bol Processor Grammars*. Mira Balaban. Understanding Music with AI, AAAI Press, pp.366-400, 1992.

[8] STEEDMAN, Mark J. A generative grammar for jazz chord sequences. *Music Perception: An Interdisciplinary Journal*, 1984, vol. 2, no 1, p. 52-77.

[9] GÓMEZ, David Alejandro Reyes. *Descripción y Aplicaciones de los Autómatas Celulares*. 2011.

[10] http://delta.cs.cinvestav.mx/~mcintosh/comun/tesismaestria/genaro/node1.html (último acceso, mayo 2017).

[11] MIRANDA, Eduardo R.; MCALPINE, K.; HOGGAR, S. Dynamical systems and applications to music composition: A research report. *Proceedings of Journees d'Informatique Musicale (JIM97). French Society for Musical Informatics (SFIM), Lyon, France*, 1997.

[12]NIERHAUS, G. *Algorithmic composition: paradigms of automated music generation.* Springer Science & Business Media. 2009.

[13] KOZA, John R. *Genetic programming: on the programming of computers by means of natural selection*. MIT press, 1992.

[14] PAPADOPOULOS, George; WIGGINS, Geraint. *A genetic algorithm for the generation of jazz melodies. Proceedings of STEP*, 1998, vol. 98.

[15] WIGGINS, Geraint, et al. Evolutionary methods for musical composition. Dai Research Paper, 1998.

[16]http://delta.cs.cinvestav.mx/~mcintosh/cellularautomata/Summer_Research_files/Arti_Ver_Inv_2011_DCM.pdf (último acceso Mayo 2017).

[17] MCCORMACK, Jon. *Grammar based music composition*. Complex systems, 1996, vol. 96, p. 321-336.

[18] DUBOIS, Roger Luke. *Applications of generative string-substitution systems in computer music*. 2003. Tesis Doctoral. Columbia University.

ANEXO II. INTRODUCCIÓN A LA HISTORIA DE LA MÚSICA Y AL ANÁLISIS MUSICAL

1. Historia de la Música

La Historia de la Música se divide en los siguientes **períodos**: Música Antigua (hasta el s. 5 d. C), Período Medieval (s. VI-XV), Período Moderno (s. XVI-XVIII). Para la comprensión de las entradas elegidas a la red solamente se van a comentar el período moderno. En la historia de la música occidental, se describen las siguientes etapas: Edad Media, Renacimiento (1420-1600), Barroco (1600-1750), Clasicismo (1720-1829), Romanticismo (1820-final s. XIX) y Música Contemporánea (1820-1907), de los cuales el Barroco, Clasicismo y Romanticismo, van a ser brevemente descritos a continuación.

Dentro de la música existen diferentes **géneros**, como la música vocal/instrumental, sinfónica/de Cámara, profana/religiosa, haciendo referencia a la utilización de la música.

Los **rasgos** más importantes de las obras musicales son: la **melodía**, el **ritmo** y la **armonía**. Existen otros rasgos como la textura, instrumentación y forma musical, pero de cara a abordar este trabajo, sólo será necesario distinguir entre los 3 primeros, ya que la forma musical, en este caso, será la **Forma Sonata**.

El origen de toda la cultura occidental se da en la **antigua Grecia**, desde el s. VII a.c., y fue la base para períodos posteriores. En el caso de la música, esto no es así, aunque es interesante comentar que la música griega era monódica, es decir, existía solamente una melodía. Además, existían 3 intervalos como los conocemos hoy en día, la octava (2:1), Quinta (3:2) y Cuarta (4:3). Estas proporciones matemáticas fueron descubiertas por Pitágoras en el s. VI a.c. La unidad básica es el tetracordo, cuatro notas con un intervalo de cuarta justa.

Posteriormente, durante la **Edad Media**, se forman instituciones eclesiásticas y es en relación a la música religiosa donde nace la teoría y notación musical. Como canto litúrgico medieval nace el **Canto Gregoriano**, canto monódico con un ritmo flexible y modal armónicamente. Este nuevo canto lleva a la creación de la teoría musical para clasificar los diferentes cantos en un sistema. Se crean así **8 modos** utilizando como criterios fundamentales la nota final y el ámbito del canto (auténtico si se añade la 4ª por encima de la 5ª y plagal si se añade por debajo). Las posibles notas finales son Re, Mi, Fa o Sol, y sus modos correspondientes *protus, deuterus, tritus* y *tetrardus*.

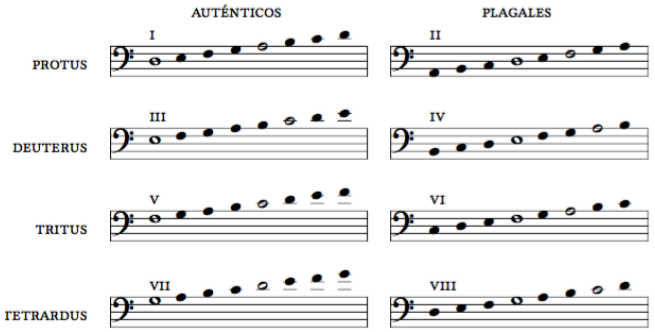


Figura 1. Sistema de los 8 modos [3]

La **notación musical** se debe a Guido d'Arezzo, que reunió varias técnicas que facilitaban la lectura a primera vista. La notación cuadrada utilizada en el Canto Gregoriano nació del sistema guidoniano.

La **polifonía** nació en el s. IX como técnica de amplificación del repertorio gregoriano. El nombre original de la polifonía era el ***organum***. Así pues el Organum Primitivo constaba del *organum* paralelo (una voz llamada organal dobla la melodía una 4ª o 5ª inferior), *organum* paralelo modificado (igual que el anterior pero finalizando al unísono) y *organum* libre (voz organal no dobla, sino se mueve por acordes de 8ª, 5ª y 4ª a la par que la melodía). Más tarde en los s. XII y XIII nace la polifonía aquitana y la escuela de Notre Dame, que tras sus influencias dieron lugar al ***Ars Antiqua*** con formas como el **conductus** y **motete**, y al ***Ars Nova*** en el s. XIV que da lugar a **la Canción**.

Ya en el **Renacimiento** se conserva el motete hasta 1490. La canción evoluciona exponencialmente en el Renacimiento Medio, y ya en el Renacimiento Tardío nace el madrigal en Italia, canciones sobre poemas no estróficos donde el compositor ponía música a cada verso. Cabe destacar compositores como **Claudio Monteverdi**, compositor de la primera ópera.

En 1600 se produce la transición del Renacimiento al Barroco, donde se desarrollan nuevas técnicas como el **bajo continuo** (la voz grave como soporte armónico o pedal), **melodía acompañada**, **contrastes** y el **estilo concertado**. El madrigal evoluciona a la cantata donde destaca A. Scarlatti, y a continuación, nace la **Ópera**.

La música instrumental del s. XVII toma una gran importancia, ya que hasta el s. XVI la música vocal era más popular que la instrumental. Las formas principales de música instrumenta en el Barroco son la **Fuga**, **Preludios** o **Toccatas**, compuestas para tecla utilizando el **contrapunto** (inversiones, imitación, etc), donde se expone un sujeto y un contrasujeto, la **Suite** de forma binaria AABB con modulación a la dominante y vuelta a tónica, la **Sonata**, sin forma fija en aquella época**,** y el **Concierto** con 3 movimientos, 1º y 3º rápidos y el 2º lento.

Los principales compositores del Barroco son Antonio Vivaldi, G. P. Teleman, G. F. Häendel y el más importante, **Johann Sebastian Bach**, destacando por *El Clave bien Temperado* o *El Arte de la Fuga* donde utiliza el contrapunto de forma magistral.

Entre el s. XVIII y XIX nació el período conocido como el **Clasicismo**, cuyos máximos exponentes fueron **W. A. Mozart** y **Beethoven**, que junto con Bach son considerados los 3 mejores compositores de todos los tiempos.

Durante el clasicismo, la forma musical se regulariza, organizándose en **semifrases**, **frases** y **períodos** de 2, 4 y 8 compases respectivamente. La estructura se puntúa en **cadencias** simplificándose el **ritmo armónico** y se crean diseños de acompañamiento como el *bajo Alberti*. Los **géneros musicales** por antonomasia son la **ópera seria** y **ópera bufa**, la **forma sonata**, **concierto** y **sinfonía** (igual que la forma sonata para orquesta), y nuevos instrumentos como el piano (evolución del clave y pianoforte) y el clarinete cobran notable importancia.

Las **formas musicales** del clasicismo son la **forma binaria** de estructura AABB (tabla 1) y la **forma sonata**.

A		B	
a	b	a	b
I → V		V → I	

Tabla 1. Forma binaria [3]

Posteriormente, en el s. XIX nace el **Romanticismo**, cuyos principales compositores son Franz Schubert, F. Chopin, R. Schumann, J. Brahms, Richard Wagner y Tchaikovsky. En el posromanticismo, que conduce a la época de la música contemporánea destacan Gustav Mahler por sus sinfonías, Richard Strauss y Claude Debussy.

En este período de la música se produce un fuerte desarrollo armónico, instrumental, orquestal, y nace el virtuosismo con Liszt en piano y Paganini en violín. El uso de **cromatismos**, **nuevos acordes** como los acordes de séptima disminuida, novena, etc, hacen que las modulaciones (cambios a otra tonalidad dentro de una misma pieza) sean más complejas. Como **formas musicales** destacan el **concierto con solista** de forma similar a la forma sonata pero con una orquesta acompañando a un solista, el **poema sinfónico** creado por Liszt y la **ópera romántica** con el estilo del *bel canto* de Rossini, Bellini y Donizetti dando gran importancia a la melodía y la **Grand Opéra** en Francia. Como compositor de óperas destaca **Giuseppe Verdi** con obras como *Rigoletto*, *La Traviata* o *Aída*, y como estilo de drama musical en Alemania, Richard Wagner que rechazaba el *bel canto*.

2. Análisis Musical

¿Qué es una tonalidad? ¿Qué es un compás? ¿Cómo se compone la música? A continuación se explicarán conceptos básicos de la teoría de la música fundamentales para leer una partitura tales como las notas musicales, el ritmo o la tonalidad.

2.1 Las Notas Musicales. La Escala

Las notas musicales se ubican en 5 líneas paralelas denominadas **pentagrama**, y reciben su nombre dentro de este pentagrama con una **clave** que se coloca al comienzo del mismo. Las notas musicales son: Do, Re, Mi, Fa, Sol, La y Si. En notación americana, muy utilizada para escribir acordes en las canciones de pop o rock, la primera nota es la A (o La) y las demás notas siguen en orden el abecedario: A – La, B – Si, C – Do, D – Re, E - Mi, F – Fa y G – Sol.

Estas notas forman lo que se denomina **escala musical**. Hay varios tipos de escalas musicales o modos ya establecidos en la antigua Grecia, pero basta con conocer la escala más utilizada en la música clásica (figura 2) y la escala cromática (figura 3). Cabe destacar que todas las notas están a una distancia de un **tono** respecto de la nota anterior salvo el Mi del Fa y el Si del Do que están a un **semitono**.

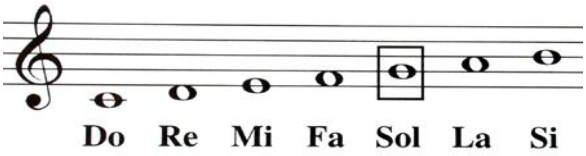


Figura 2. Escala musical

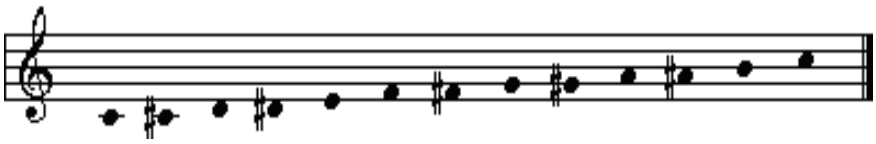


Figura 3. Escala cromática

Las notas pueden ser alteradas, es decir, se puede modificar su tono con 3 partículas denominadas **alteraciones**: sostenido, becuadro y bemol (figura 4). El sostenido aumenta un semitono, el bemol lo baja y el becuadro anula los 2 anteriores. Estas alteraciones se pueden encontrar justo a la derecha de la clave, lo que da información de la **tonalidad** en la que está la obra, que da lugar a la **armadura**. Si por ejemplo hay un sostenido en la 5ª línea del pentagrama (nota Fa para clave de Sol), cada vez que aparezca un Fa en el pentagrama estará alterado, a no ser que tenga a la izquierda de la propia nota un becuadro que anulará su efecto. Las alteraciones que aparecen a lo largo de la obra y actúan solamente sobre la nota a la que acompañan se denominan **alteraciones accidentales**.

Sostenido	*Bemol*	*Doble Sostenido*	*Doble Bemol*	*Becuadro*
♯	♭	𝄪	♭♭	♮

Tabla 2. Alteraciones

2.2 La Clave

El símbolo al inicio de cada compás se conoce como clave. La clave sitúa todas las notas en el compás. La **clave de Sol**, que se puede observar en las figuras 2 y 3, sitúa la nota Sol en la 2ª línea del pentagrama, y a partir de ahí se sitúan las demás siguiendo el orden descrito anteriormente. En esta clave leen los violines, piano, clarinete, etc. Existen otras claves como la clave de Do o la clave de Fa que sitúan el Do y el Fa respectivamente en el pentagrama. En el caso de la clave de Fa, la nota Fa se situaría en la 4ª línea, y es la clave en la que leen los violoncellos. Una clave es ascendente cuando tiene una dirección hacia arriba como en las figuras 2 y 3, y descendente si va hacia abajo.

2.3 Las Tonalidades

En armonía, que es la asociación de varios sonidos simultáneos para la formación de acordes, se construyen acordes de acuerdo a una tonalidad, que es la base de la obra musical. Las **tonalidades** pueden variar en una misma obra produciéndose una **modulación**. En cada tonalidad se alteran más o menos notas. Hay 2 tonalidades en las que se alteran las mismas notas. Una de ellas será mayor y la otra menor, siendo relativas. En la figura 4 se muestran las tonalidades mayores por fuera, y sus relativos menores por dentro.

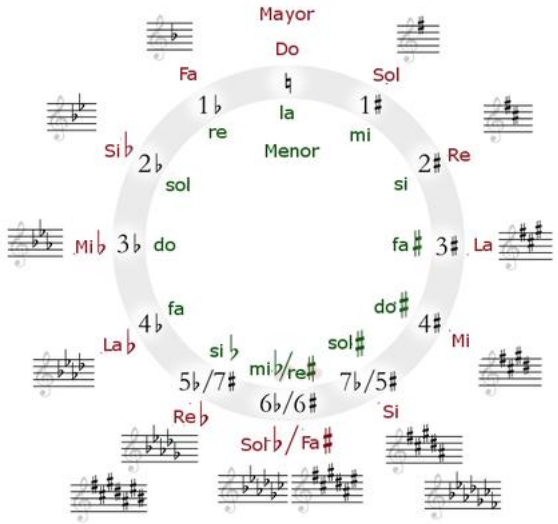


Figura 4. Tonalidades

El orden en el que se alteran las notas para los sostenidos es: Fa, Do, Sol, Re, La, Mi y Si, y para los bemoles el orden contrario: Si, Mi, La, Re, Sol, Do y Fa. Por ejemplo, para las tonalidades Re Mayor y su relativo Si menor, se alterarán 2 notas con sostenidos como muestra la figura 4 en el orden descrito anteriormente: Fa y Do.

2.4 Acordes

Un acorde es un conjunto de 3 o más notas que forman una entidad al superponerse verticalmente.

Los acordes se construyen en base a la tonalidad correspondiente en tríadas. El acorde de 5ª o dominante es junto a la tónica 1er grado el más importante. El acorde más común es el perfecto. Este acorde puede ser Mayor o menor. El acorde Perfecto Mayor está formado por el 1er grado del acorde, el 3º con una distancia sobre el 1er grado de 3ª Mayor (2 tonos) y el 5º en una distancia de 5ª Justa (3 tonos y 1 semitono sobre el 1er grado). El acorde perfecto Mayor de Do se muestra en la figura 5.

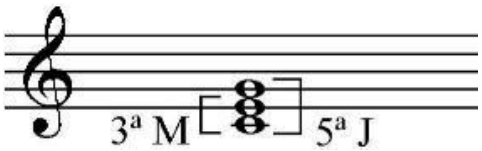


Figura 5. Acorde Perfecto Mayor de Do

2.5 Cadencias

Una cadencia es un conjunto de acordes en los que termina una frase musical en un grado determinado como el primero (tónica), el 5º (dominante), etc. Existen distintos tipos de cadencias como la cadencia perfecta, plagal o semicadencia.

2.6 Ritmo

Las obras musicales están estructuradas en base a un **compás**. Un compás es una entidad musical compuesta por varias unidades de tiempo denominadas **figuras**, que se organizan en grupos.

En la notación occidental, los compases aparecen representados al principio del pentagrama a la derecha de la clave y alteraciones que determinan la tonalidad. El compás consta de 2 números. El denominador indica la figura equivalente a un tiempo en dicho compás y el numerador cuántas de esas figuras caben en un compás, de acuerdo con la figura 6.

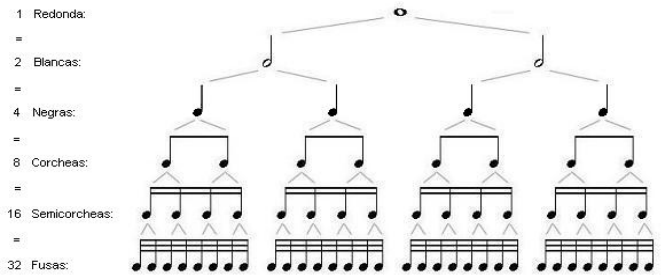


Figura 6. Figuras

Por ejemplo, en un compás de 2/4, habrá 2 negras por compás ya que el 4 del denominador indica que el tiempo es de negra y el 2, que hay 2 por compás.

3. La Forma Sonata

3.1 Estructura

La **forma sonata** [3, p.101] es la forma musical más utilizada en el mundo de la música clásica. Esta estructura musical pertenece principalmente al primer movimiento de los 4 movimientos (tabla 3).

I	Allegro	Forme de sonata
II	Lento	Forma de sonata Forma de sonata sin desarrollo Tema con variaciones Forma ternaria
III	Minueto	Minueto o scherzo con trío
IV	Finale	Forma de sonata Rondó Rondó-sonata

Tabla 3. Estructura musical más habitual

En música clásica, los temas y frases que se definen posteriormente se componen a partir de **motivos**, que son fragmentos de 2 o 3 compases que caracterizan al resto del tema donde se encuentra toda la información rítmica y melódica, por tanto, el motivo es el núcleo a partir del cual se compone el resto de la música y se encuentra al comienzo de cada tema.

La forma sonata se divide en 3 partes: exposición, desarrollo y reexposición. En la **exposición** se exponen 2 temas[4], el **Tema A** en tónica, rítmico, es decir, un tema en el que lo más importante es el ritmo, no la armonía, una **transición** o puente modulante[5] a la dominante donde está el **Tema B**, que a su vez consta de los temas b_1, b_2 y b_3, más melódicos que el tema A, y finaliza con un **coda**[6]. Cada uno de estos temas tiene al final una pequeña cadencia que indica su final y el principio del tema siguiente.

En el **desarrollo** se utilizan los temas A y B y se desarrollan, modulándolos a otras tonalidades incluso incluyendo nuevos elementos no utilizados anteriormente en la obra. El desarrollo concluye con un puente modulante que vuelve a la tonalidad de inicio del desarrollo.

En la **reexposición**, se repite la exposición modificándola, ya que los temas A y B aparecen en tónica, con otra transición. Por último, la **coda** es la sección que concluye la estructura, repite la coda de la exposición en la tonalidad principal. Esta forma de componer se utiliza tanto en sonatas, como sinfonías, conciertos, etc.

Cabe destacar que los Temas A y B se construyen a partir de motivos de 1 o 2 compases que se desarrollan formando cada tema. Para identificar los motivos basta con identificar el ritmo que se repite a lo largo del tema.

Un resumen de esta forma sonata se expone en la tabla 4.

EXPOSICIÓN	DESARROLLO	REEXPOSICIÓN
• **A** – Tema principal rítmico en tónica • **Transición** • **B** – Temas b_1, b_2 y b_3 en otra tonalidad	• Elaboración, desarrollo de la temas A y B • (pueden aparecer elementos nuevos)	• **A** – Tema principal rítmico en tónica • **Transición** modificada • **B** – en la tonalidad principal • **Coda**, Final

Tabla 4. Forma Sonata

[4] Desarrollado a partir de un motivo, es un fragmento de música en el que predomina un ritmo.

[5] Unión entre 2 frases musicales. En el caso de una forma sonata, consta de varios compases en los que se pasa de la tonalidad (principal en este caso) a otra distinta.

[6] Cadencia expandida.

3.2 Ejemplo. Sonata nº1 op.10 para piano de Beethoven

Como ejemplo de análisis de una forma sonata se muestra el primer movimiento de la Sonata de Beethoven para piano nº1.

El Tema A de esta sonata está compuesto a partir del motivo de la figura 5.

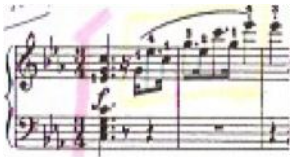
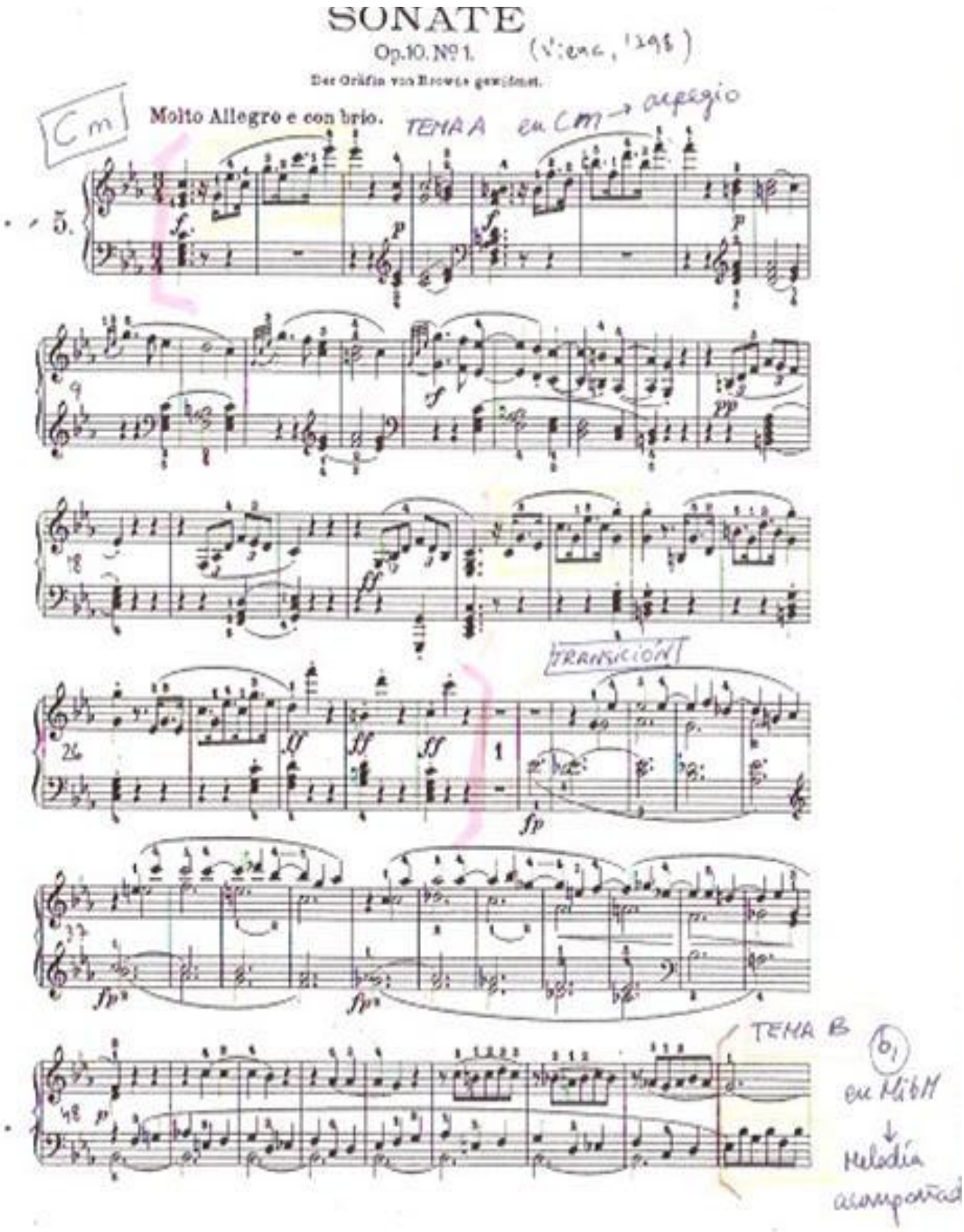


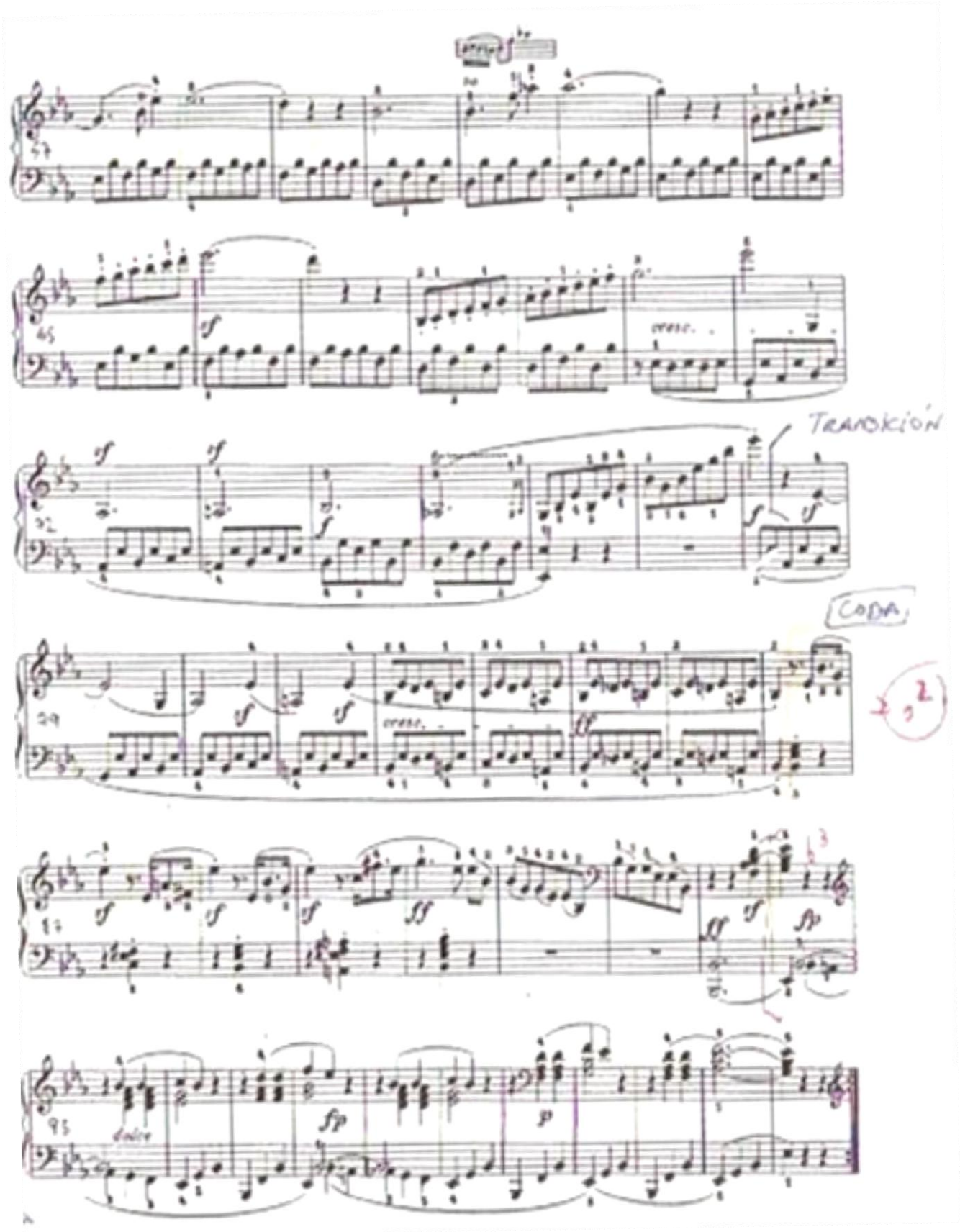
Figura 5. Motivo del Tema A

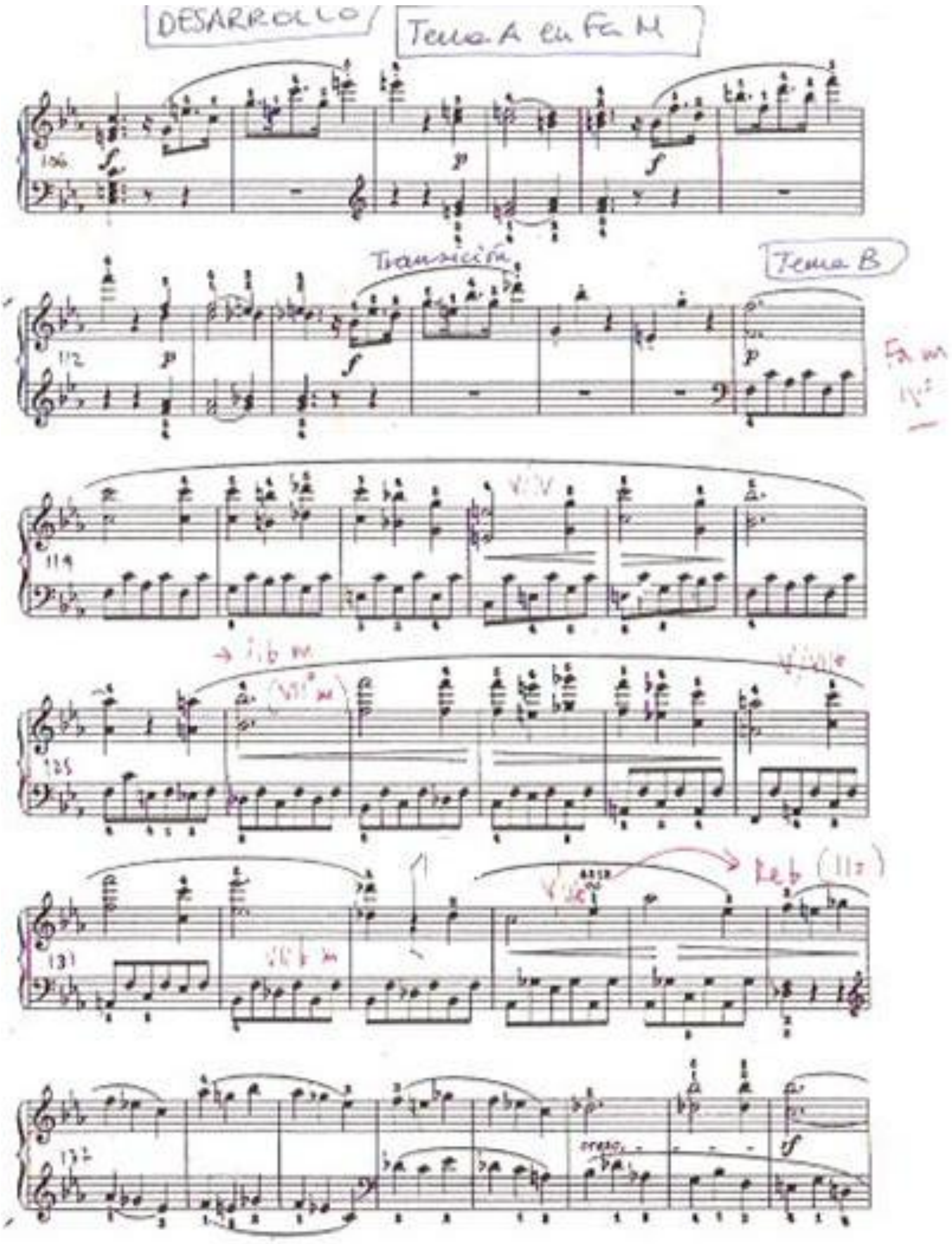
El Tema B se puede identificar por la voz del bajo en clave de Fa, que cambia respecto del tema A. En la figura 6 se muestra la obra analizada y en la tabla 4, el esquema de la obra.

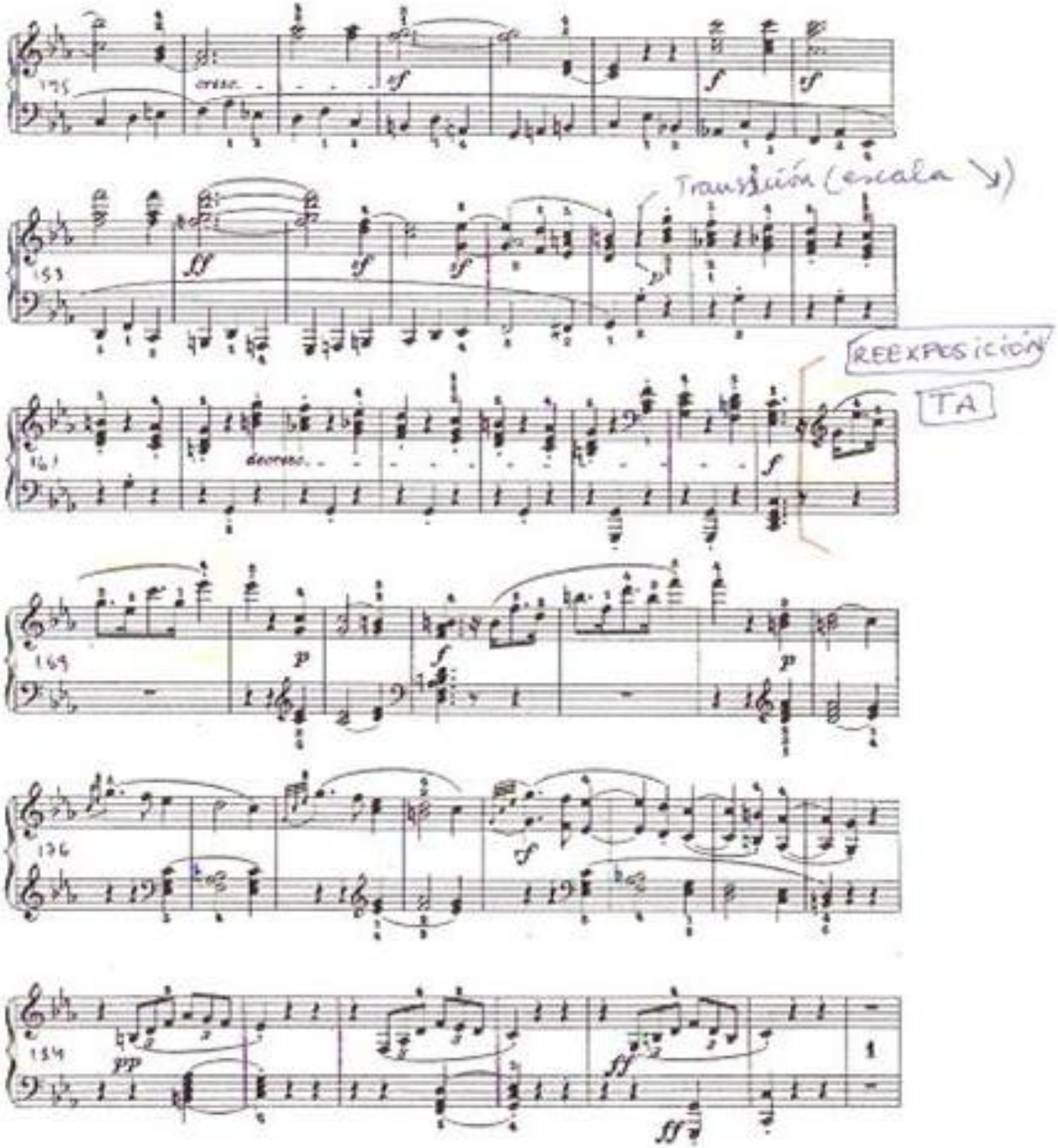
<table>
<tr><td colspan="4">EXPOSICIÓN</td><td>DESARROLLO</td><td>REEXPOSICIÓN</td></tr>
<tr><td colspan="4">compás 1 - 105</td><td>compás 106 - 168</td><td>compás 168 - 284</td></tr>
<tr><td>Tema A</td><td>Trans.</td><td>Tema B</td><td>Coda</td><td></td><td></td></tr>
<tr><td>c. 1-30</td><td>c. 31-55</td><td>c. 56-63</td><td>c. 64-105</td><td></td><td></td></tr>
</table>

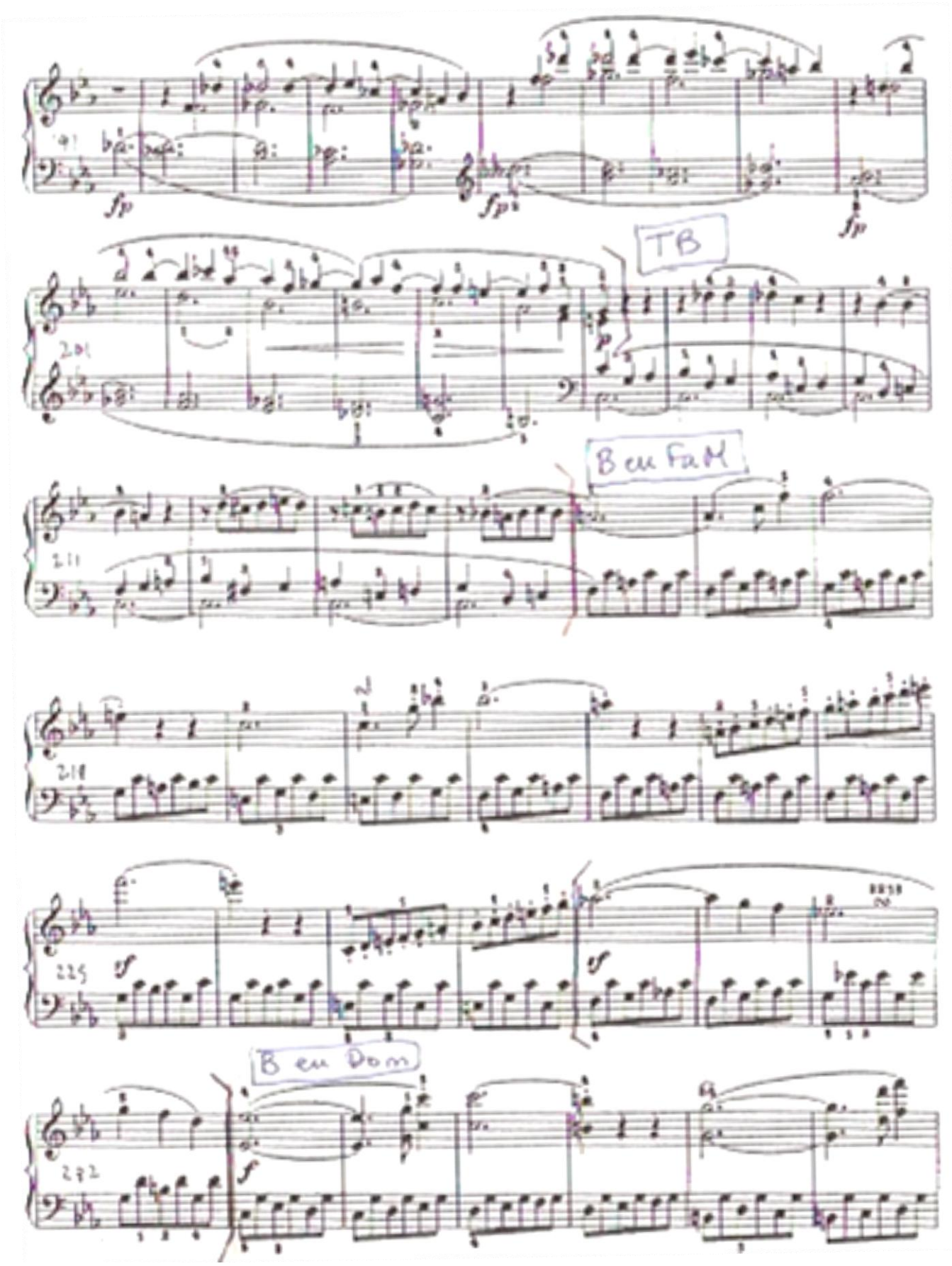
Tabla 4. Estructura de la Sonata nº1 op.10 para piano de Beethoven, Molto Allegro

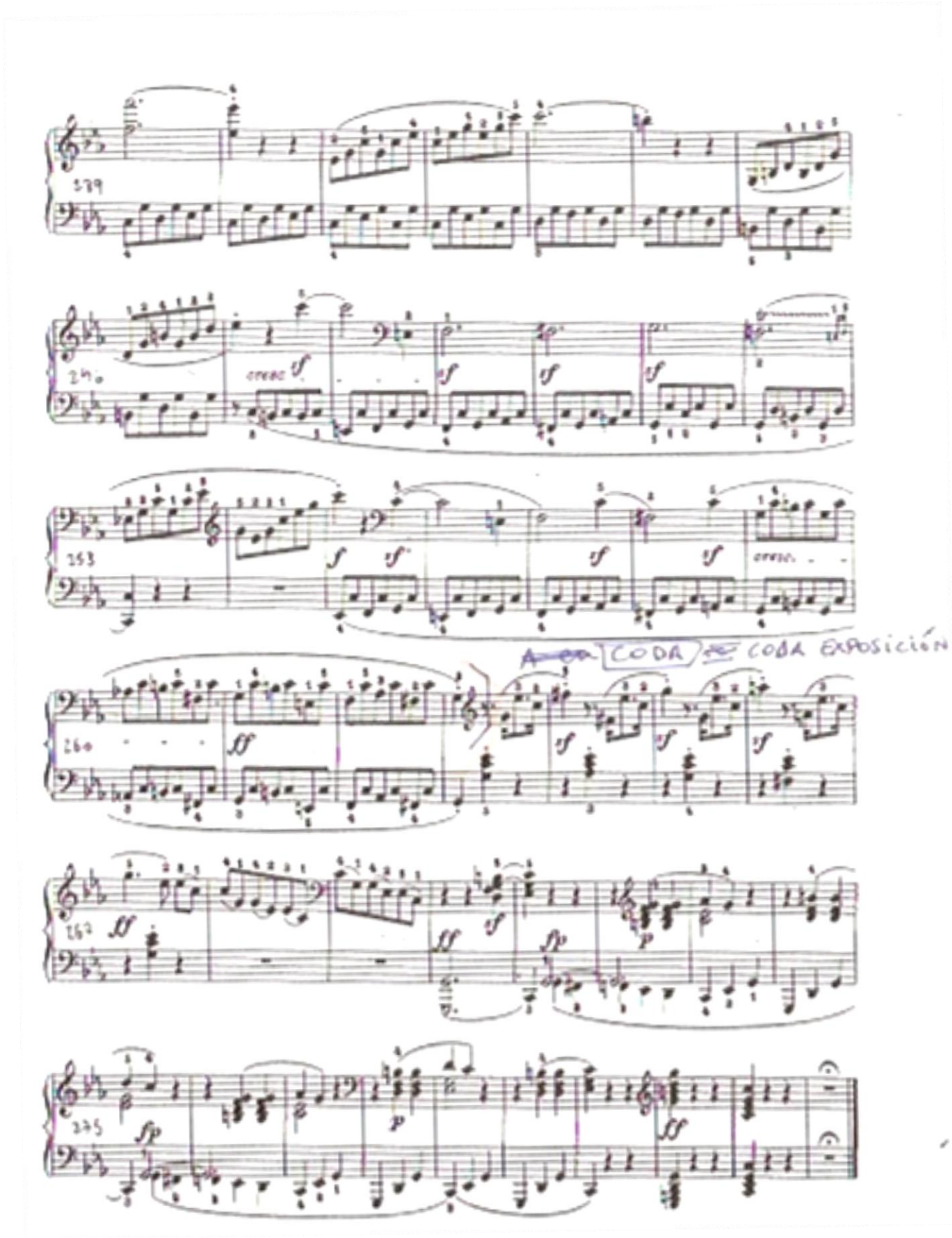

SONATE
Op.10. Nº 1.
Molto Allegro e con brio.
Cm
TEMA A en Cm → arpegio
TRANSICIÓN
TEMA B
6₁
en Mib M
melodía acompañada


TRANSICIÓN
CODA


DESARROLLO
Tema A en Fa M
Transición
Tema B


Transición (escala ↘)
REEXPOSICIÓN
TA


TB
B en Fa M
B en Dom


CODA
CODA EXPOSICIÓN

4. Referencias

[1] J. GROUT, Donald. V. PALISCA, Claude. Historia de la Música Occidental. Ed. Alianza, vol. 1

[2] J. GROUT, Donald. V. PALISCA, Claude. Historia de la Música Occidental. Ed. Alianza, vol. 2

[3] GUERRERO, Francisco. Historia de la música. *CPM Curso*, 2014-2015

ANEXO III. DEEP LEARNING Y BIG DATA

1. Introducción

El Deep Learning y Big Data son dos conceptos de estudio en la ciencia de datos. El Big Data ha cobrado una importancia muy elevada en numerosas organizaciones públicas y privadas, y puede contener datos de ciber seguridad, marketing, etc. Los algoritmos Deep Learning extraen un elevado nivel de abstracciones complejas como representación de datos a través de un proceso de aprendizaje jerárquico. Una de las principales ventajas del Deep Learning es el análisis y representación de grandes cantidades de datos sin supervisión, lo que hace del Deep Learning una herramienta clave para el análisis del Big Data. Con estos algoritmos se pueden resolver problemas introducidos en el análisis del Big Data como la extracción de patrones complejos de grandes volúmenes de datos, marcado rápido de datos, e investigar nuevos desafíos como la escalabilidad de modelos.

2. Deep Learning

El concepto de **aprendizaje automático** nació de las manos de Arthur Samuel, quién está considerado el padre del aprendizaje automático. Samuel creó en 1956 el primer programa que aprendía a jugar a las damas. Para ello, hizo jugar al programa contra sí mismo miles de veces para que aprendiese a jugar. Más tarde, en 1962, el programa ganó la competición estatal de Connecticut.

El primer gran éxito del aprendizaje automático fue Google. Google permite encontrar información con un algoritmo basado en el aprendizaje automático. Otros servicios como Amazon, Netflix, etc, sugieren artículos o películas, y redes sociales como Facebook o LinkedIn, sugieren posibles amigos que puedes tener.

Geoffrey Hinton de la Universidad de Toronto, junto a su equipo, para un proyecto de Kaggle, hizo un algoritmo de aprendizaje automático para el descubrimiento de medicamentos en tan sólo 2 semanas. El algoritmo consiguió batir a la comunidad internacional, a pesar de que en el equipo de Hinton no había nadie experto en química o biología. El tipo de algoritmo utilizado por Hinton y su equipo fue un algoritmo basado en el aprendizaje profundo o ***Deep Learning***.

El aprendizaje profundo o *Deep Learning* es un algoritmo basado en el cerebro humano, y como resultado, no tiene limitaciones teóricas en lo que puede llegar a hacer. Cuantos más datos y mayor es el tiempo de cálculo, mejor funciona. El *Deep Learning* se basa en redes neuronales que aprenden por un entrenamiento en base a una entrada suministrada. Cuanto mayor es esta entrada, para lo cual se definirá a continuación el concepto de Big Data, mejor será el aprendizaje pero también más costoso.

Richard Rashid, mediante un experimento demuestra que las computadoras pueden escuchar y comprender. Dicho experimento recopila información de hablantes de chino, y mediante un sistema de conversión texto-voz lo convierte en lengua oral. Después, graba una hora de su propia voz y la usa para modular el texto estándar de conversión texto-voz para que suene como su propia voz.

Las computadoras también han aprendido a ver y leer. Google diseñó un algoritmo que pudiera ver vídeos de youtube, y procesó la información en 1600 computadoras al mes. Las máquinas aprendieron a reconocer personas y gatos solamente viendo vídeos. También, dada una oración, entienden que expresa sentimientos negativos, por tanto, entienden qué significan las frases y qué están diciendo. Además, entienden lo que oyen y pueden escribir, o describir imágenes.

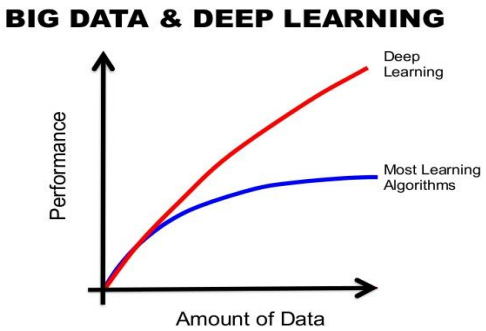


Figura 1. Big Data y Deep Learning

3. Big Data

3.1. Introducción. ¿Qué es el Big Data?

El término "***Big Data***" se refiere a aquellos activos de información que se caracterizan por su alto volumen, velocidad, variedad y veracidad, que demandan soluciones innovadoras y eficientes de procesado para la mejora del conocimiento y toma de decisiones en las organizaciones.

El objetivo fundamental del *Big Data* es dotar la creación de una infraestructura tecnológica con la finalidad de poder almacenar, tratar y analizar de manera económica, rápida y flexible la gran cantidad de datos que se generan diariamente, para ello es necesario el desarrollo y la implantación tanto de hardware como de software específicos que gestionen todos estos datos con el objetivo de extraer valor para obtener información útil para nuestros objetivos o negocios. El *Big Data* puede aplicarse tanto en empresas multinacionales como Google o Facebook así como en pequeñas empresas.

El concepto de *Big Data* aplica para toda aquella información que no puede ser procesada o analizada utilizando procesos o herramientas tradicionales

3.2. Tipos de Datos en Big Data

Existe una amplia variedad de tipos de datos a analizar [1]. Una posible clasificación es la mostrada en la tabla 1, clasificación **por origen**, aunque estas categorías pueden extenderse con el avance tecnológico.

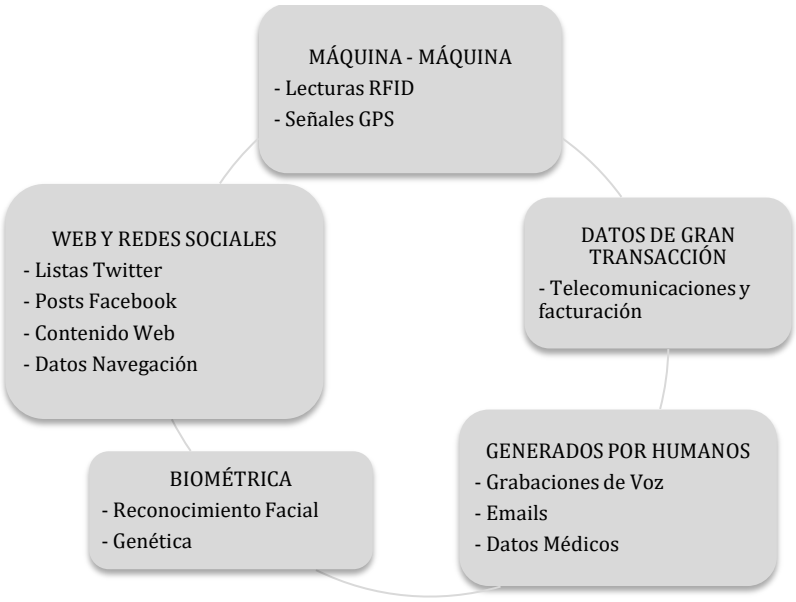


Figura 2. Tipos de datos Big Data por origen

Otra clasificación puede ser **por categorías**, donde se pueden distinguir 2 categorías principales:

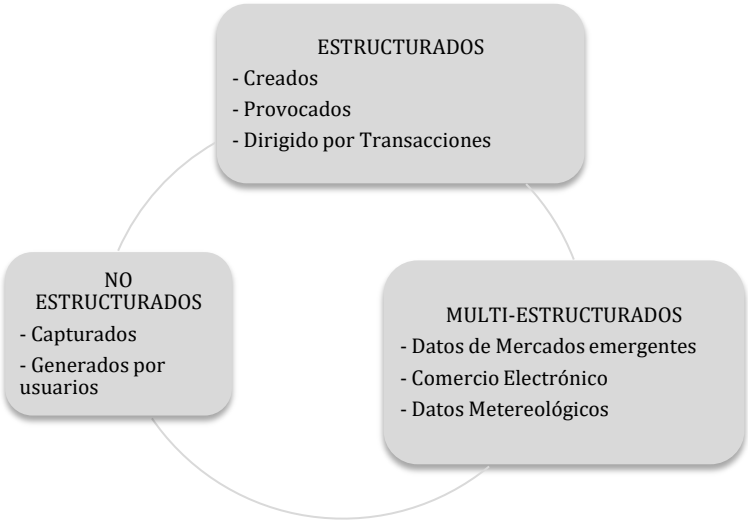


Figura 3. Tipos de datos Big Data por categorías

3.3. Componentes de una Plataforma Big Data

Para poder analizar estas grandes cantidades de datos se precisa de una plataforma [2] que además sea segura. La plataforma líder en términos de seguridad es ***Hadoop***, que es un *framework* de código abierto.

Hadoop está basado en el proyecto Google File System (GFS) y en *Map Reduce*. Está compuesto de *Hadoop Distributed File System* (HDFS), *Hadoop MapReduce* (figura 8) y *Hadoop Common*.

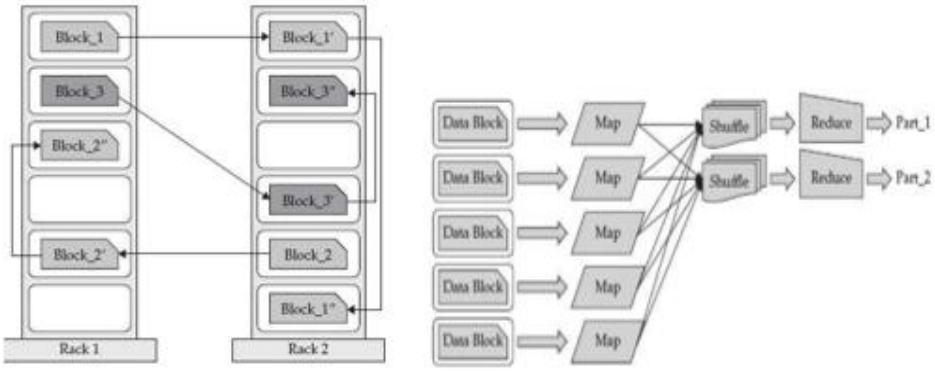


Figura 4. HDFS y Hadoop MapReduce

HDFS divide los datos en el cúster en pequeños bloques.

Hadoop MapReduce es el núcleo de Hadoop y se refiere a dos procesos ejecutados por el Hadoop. Un proceso, *map*, que toma un conjunto de datos convirtióndolo en otro conjunto de datos, y el proceso reduce que obtiene la salida de *map* como datos de entrada y combina las *tuplas* o elementos individuales en un conjunto más pequeño.

Hadoop Common son librerías que soportan subproyectos de *Hadoop*. Otros proyectos son Avro, Cassandra, Chukwa, Flume, HBase, Hive, Jaql, Lucene, Oozie, Pig o ZooKeeper.

4. Aplicaciones del Deep Learning en el Análisis del Big Data

Los algoritmos de aprendizaje profundo [3] extraen representaciones abstractas significativas de los datos en bruto, a través de un enfoque de aprendizaje multi-nivel jerárquico, donde en un nivel más alto las representaciones más abstractas y complejas son aprendidas en base a conceptos menos abstractos y representaciones en los niveles más bajos en la jerarquía de aprendizaje.

Entre las características de las representaciones abstractas se incluyen los **modelos lineales simples**, que pueden trabajar con los conocimientos obtenidos a partir de representaciones de datos más complejos, una mayor **automatización de la extracción de representación de datos** a partir de datos sin supervisión y el **conocimiento relacional y semántico**, que se obtiene en niveles superiores de abstracción.

Teniendo en cuenta las características del *Big Data*: volumen, velocidad, variedad y veracidad, las arquitecturas de aprendizaje profundo se adaptan mejor a cuestiones relacionadas con el **volúmen** y la **variedad** del *Big Data*. El *Deep Learning* explota la disponibilidad de grandes cantidades de datos y el análisis de los mismos representados en diferentes formatos, minimizando la necesidad de personas expertas para extraer características de todos los nuevos tipos de datos obtenidos observados en un gran volumen de datos.

5. Problemas del Deep Learning en el Análisis del Big Data

Ciertas áreas donde el aprendizaje profundo requiere un mayor análisis, como el aprendizaje con flujo de datos, son los datos de alta dimensión, la escalabilidad de modelos y la computación distribuida.
En **datos de alta dimensión**, como imágenes, etc, debido al lento proceso de aprendizaje los algoritmos de aprendizaje profundo pueden bloquearse cuando se trabaja con grandes cantidades de datos, ya que cuanta más información, más complicado es el aprendizaje. Los autoencoders de eliminación de ruido marginados (MSDA) son una solución a este problema.

Los **modelos a gran escala** plantean desafíos como la elección del número óptimo de parámetros del modelo y la mejora de su funcionalidad computacional. La adaptación de dominio y flujo de datos son problemas generados por grandes volúmenes de datos que necesitan innovación en modelos a gran escala.

6. Futuro del Deep Learning en el Análisis del Big Data

Problemas como la utilización de todo el corpus de entrada de grandes volúmenes de datos disponibles al analizar los datos con algoritmos de aprendizaje profundo, y la elección de qué criterios son necesarios y deben definirse para permitir que las representaciones de los datos extraídos proporcionen significado semántico útil para el *Big Data*, son retos que deben resolverse y que todavía están pendientes en el análisis del *Big Data* con *Deep Learning*.

7. Referencias

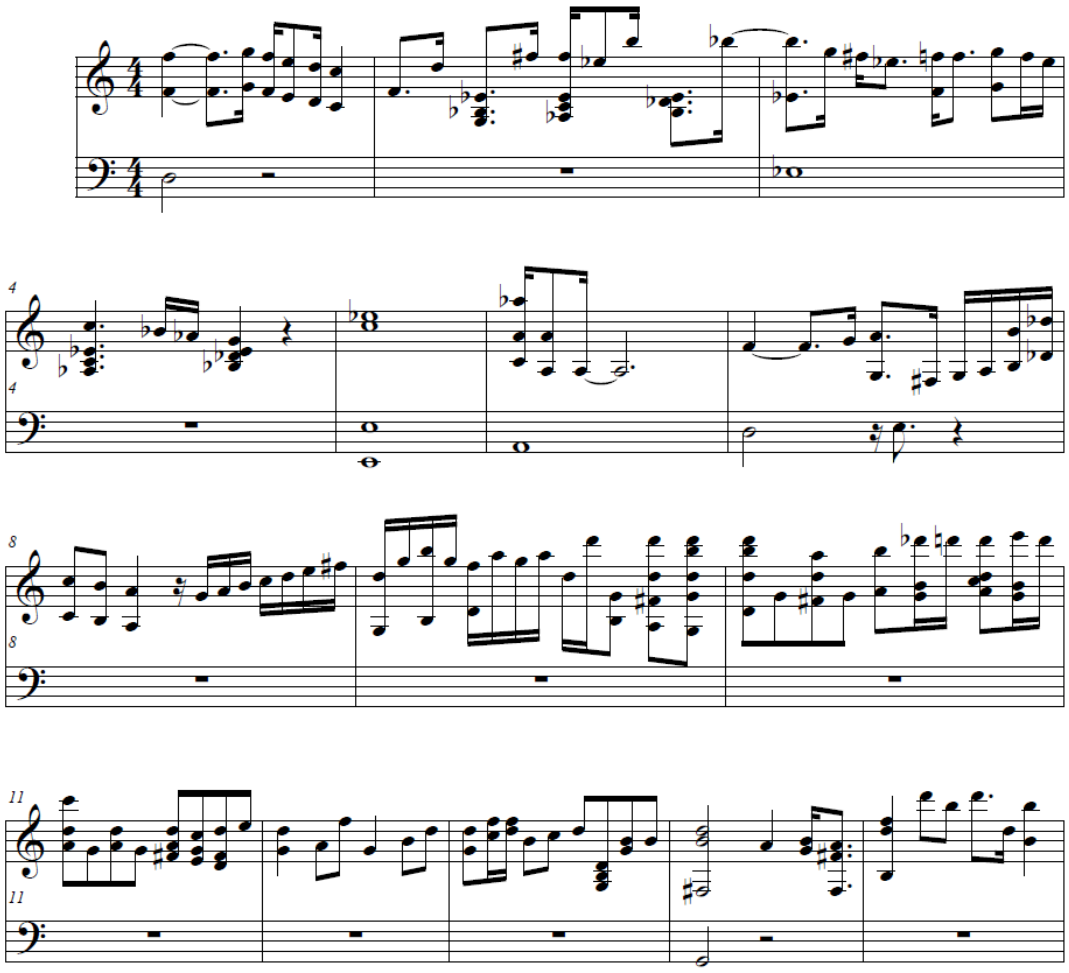
[1] http://www.netmind.es/knowledge-center/tipos-de-datos-en-big-data (último acceso Mayo 2017).

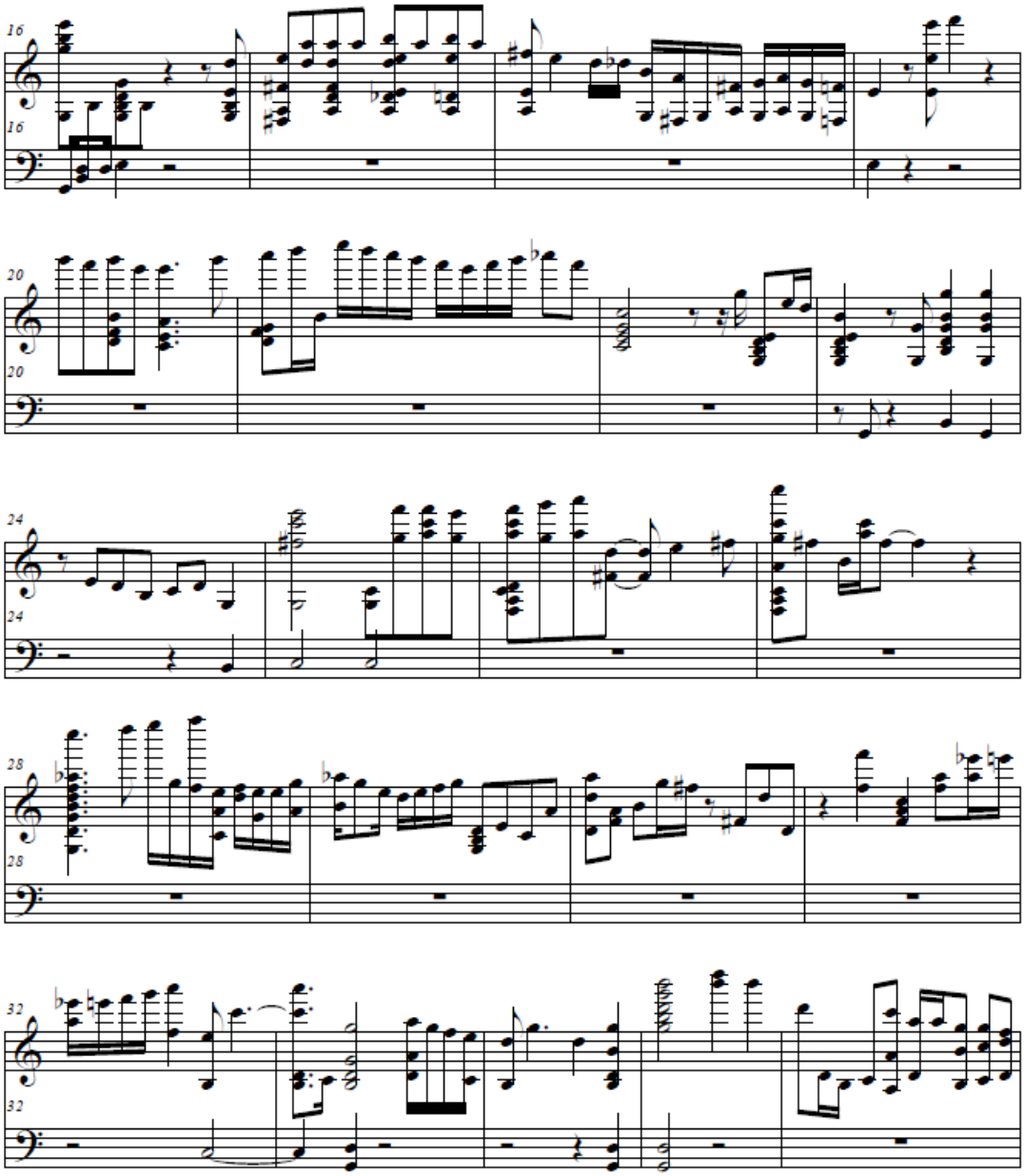
[2] https://www.ibm.com/developerworks/ssa/local/im/que-es-big-data (último acceso Mayo 2017).

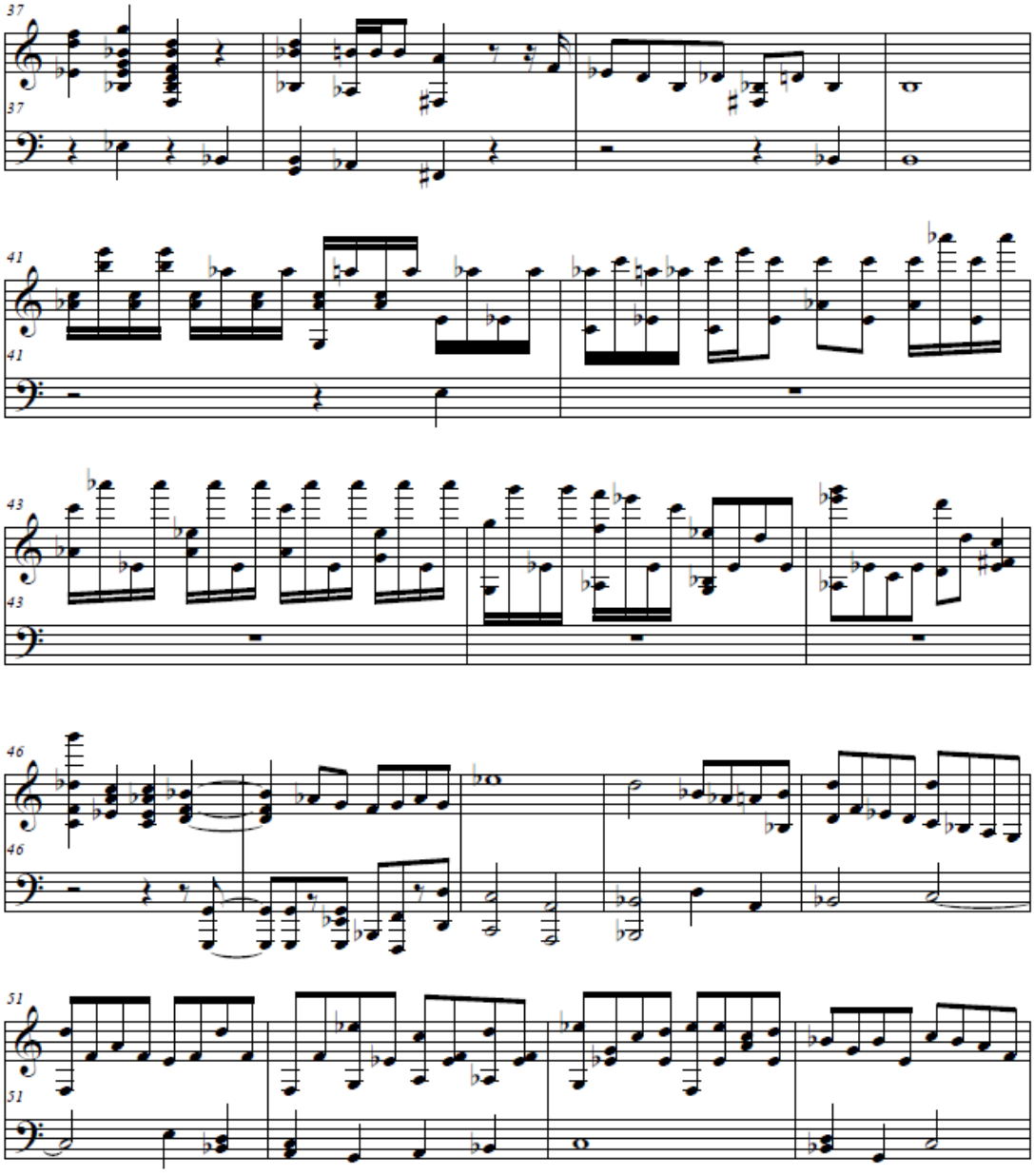
[3] https://journalofbigdata.springeropen.com/articles/10.1186/s40537-014-0007-7 (último acceso Mayo 2017).

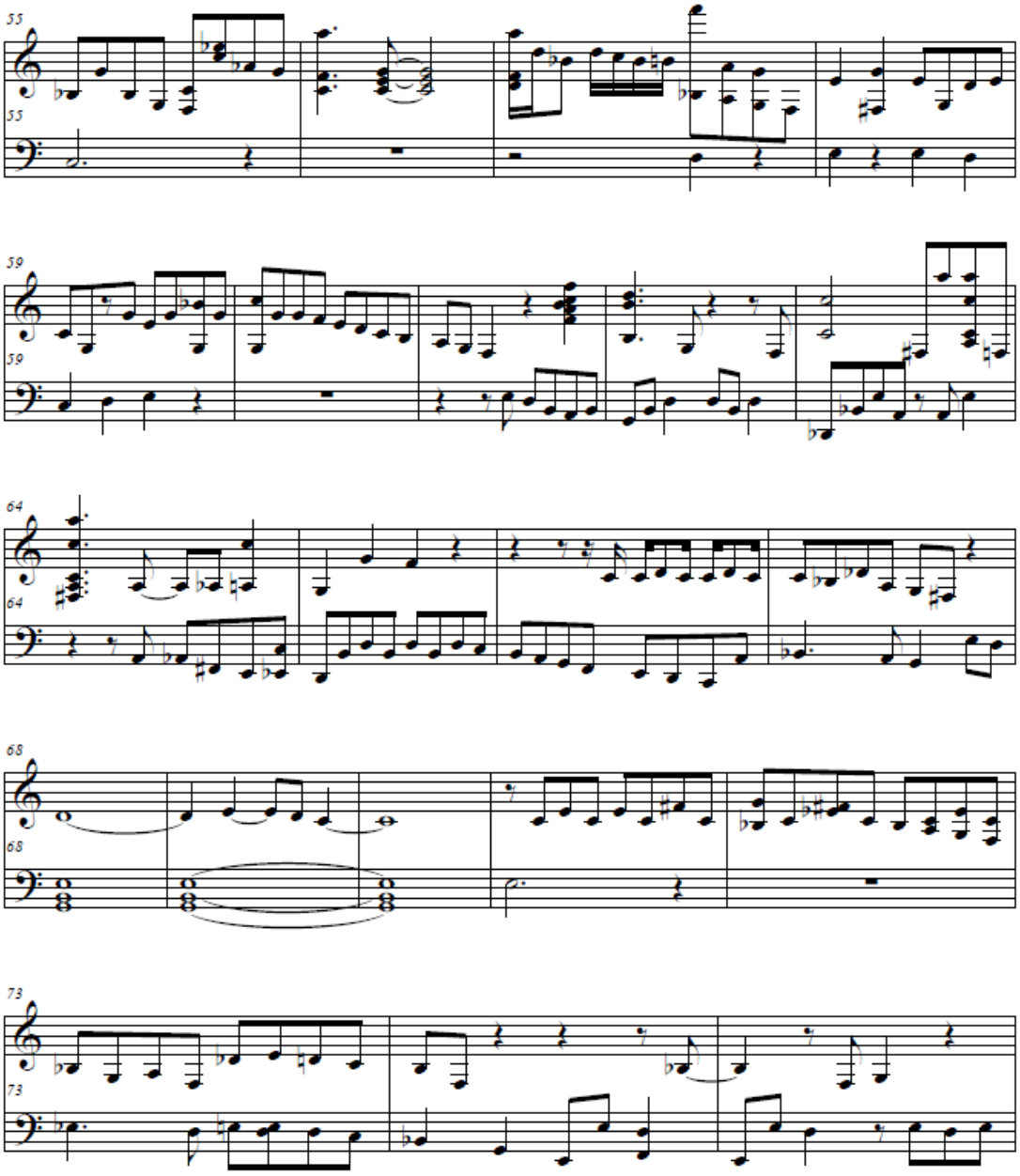
ANEXO IV. PARTITURA DE LA OBRA RESULTANTE DEL ENTRENEMIENTO DE LA RED

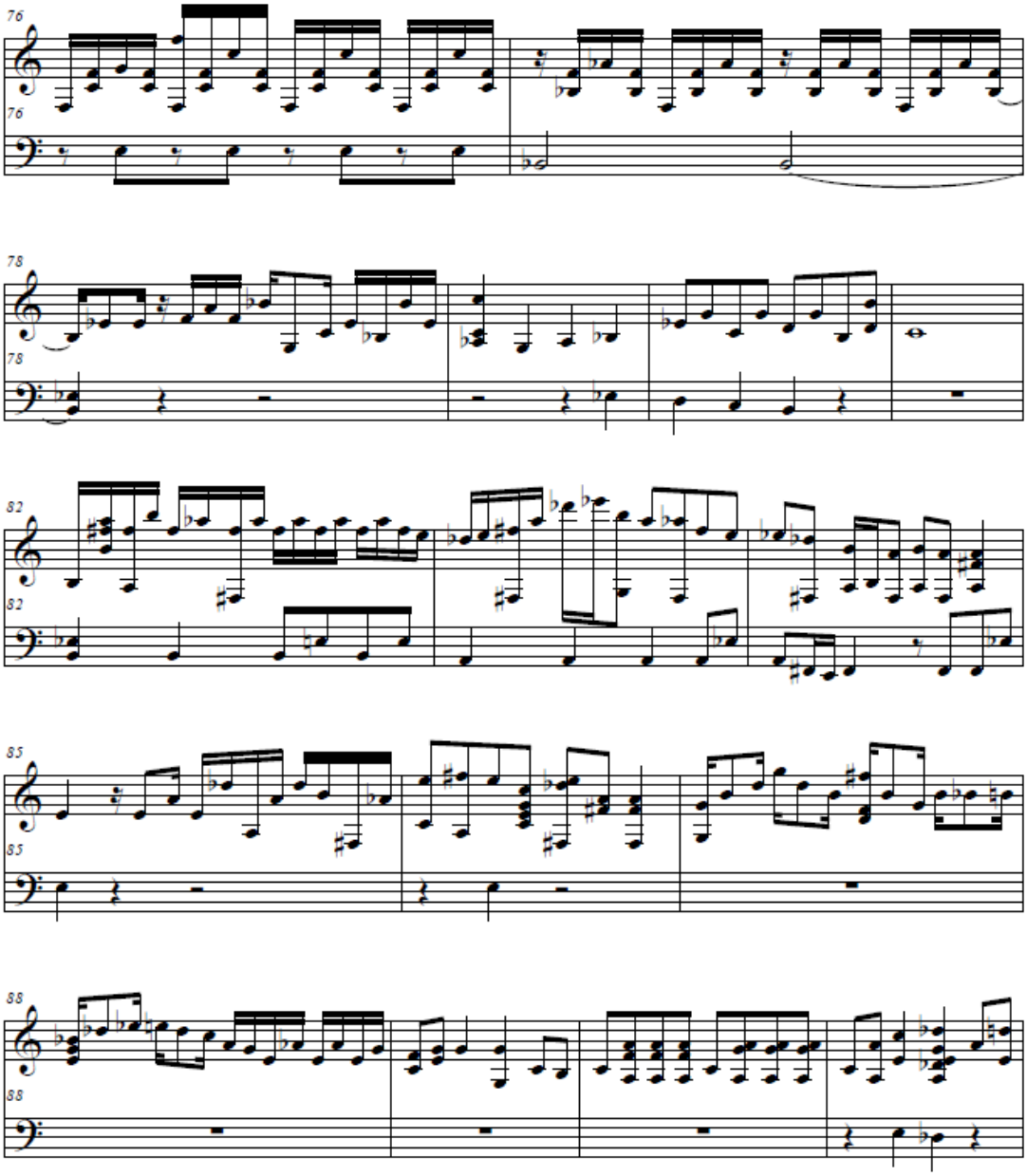
Carlos Hernández Oliván

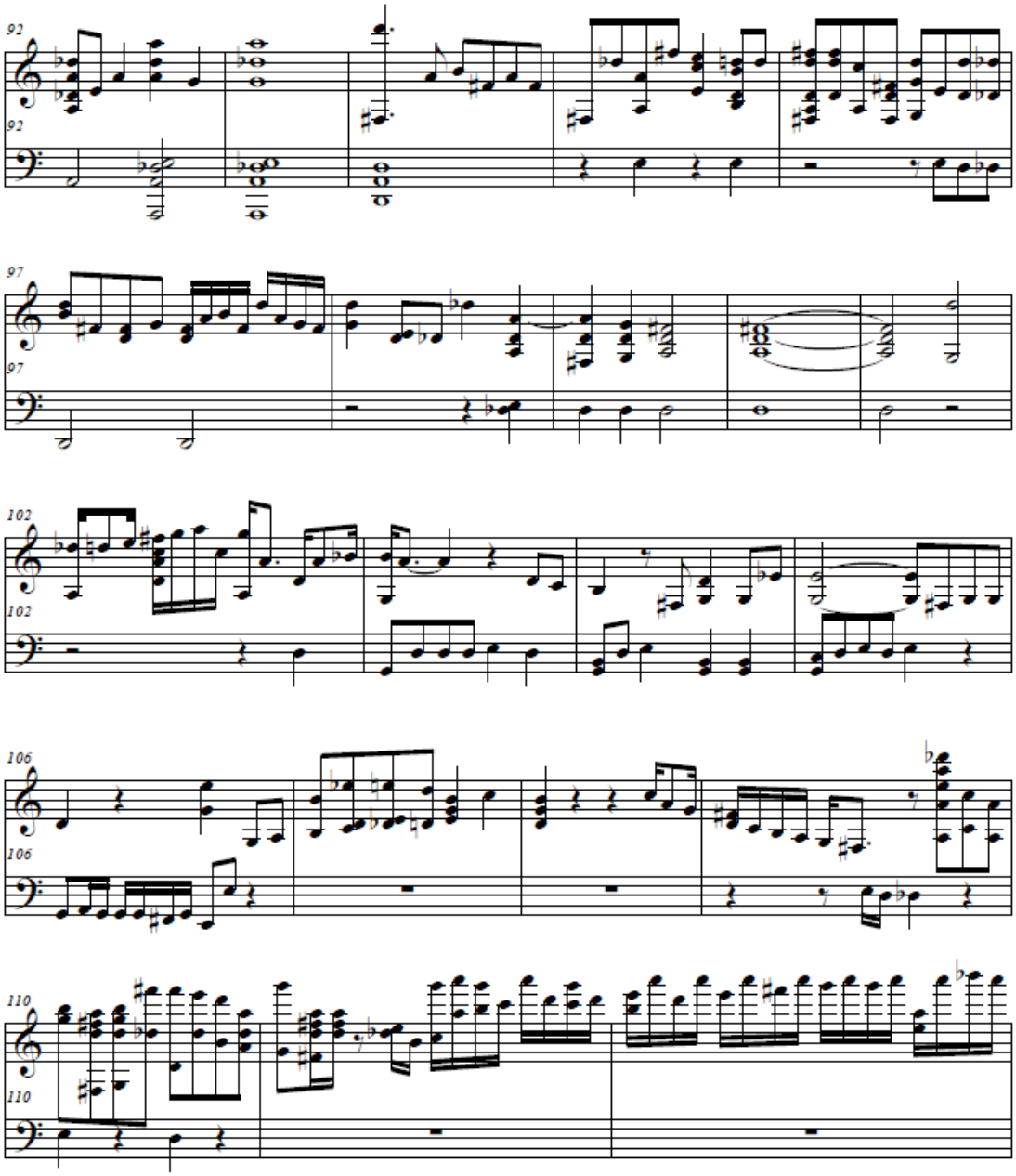


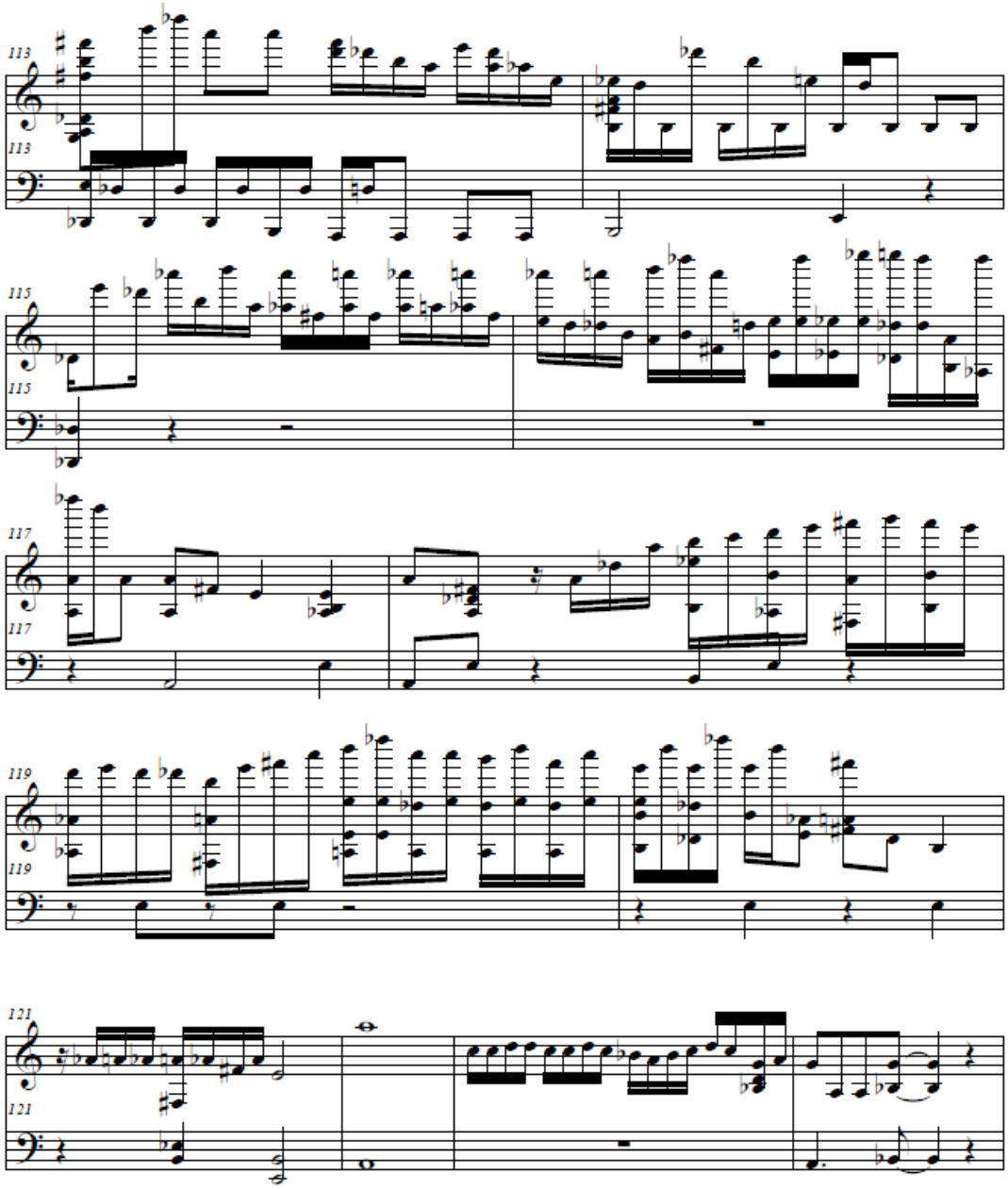

16
16
20
20
24
24
28
28
32
32

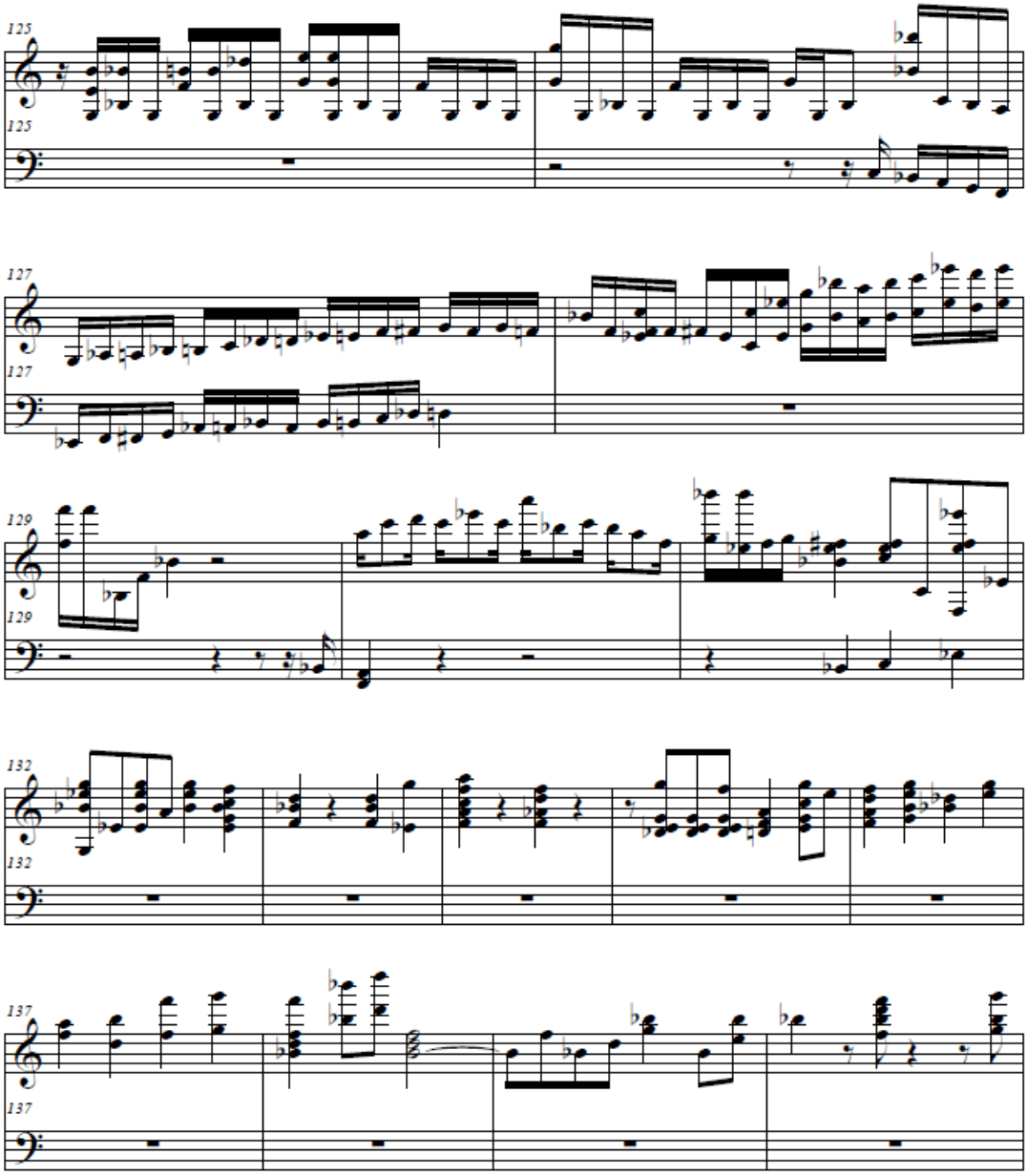

37
37
41
41
43
43
46
46
51
51

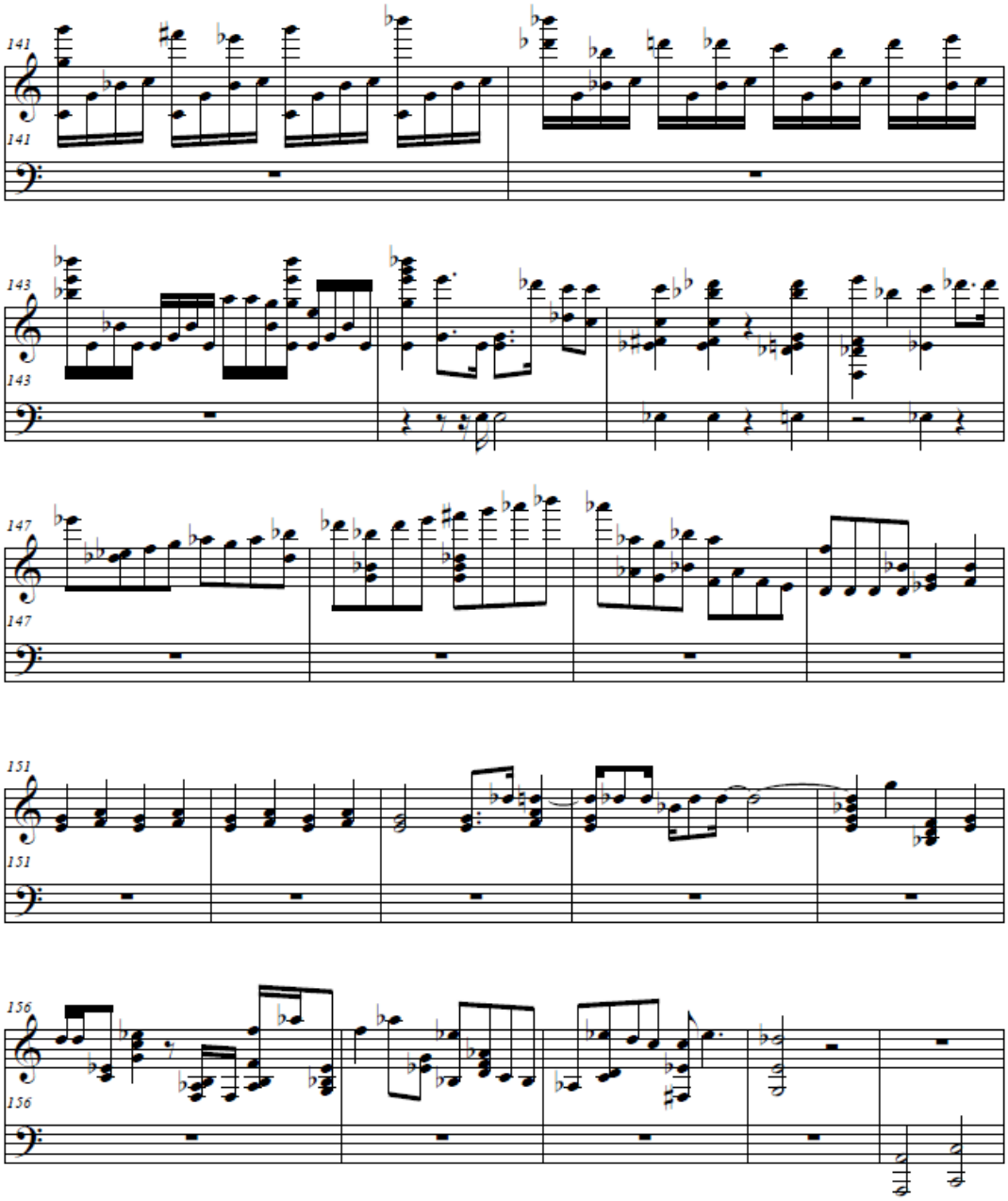

55
55
59
59
64
64
68
68
73
73

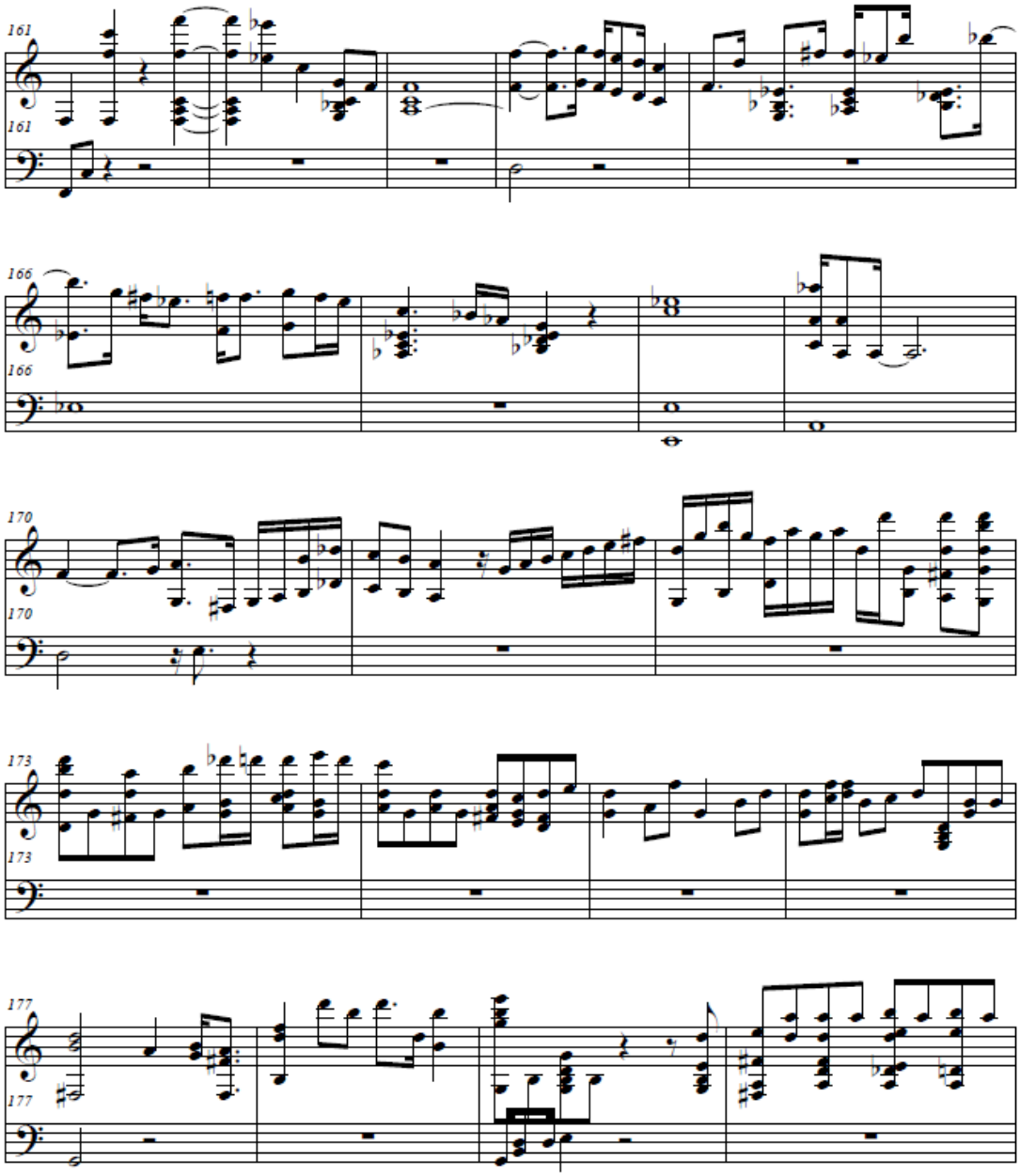

76
76
78
78
82
82
85
85
88
88

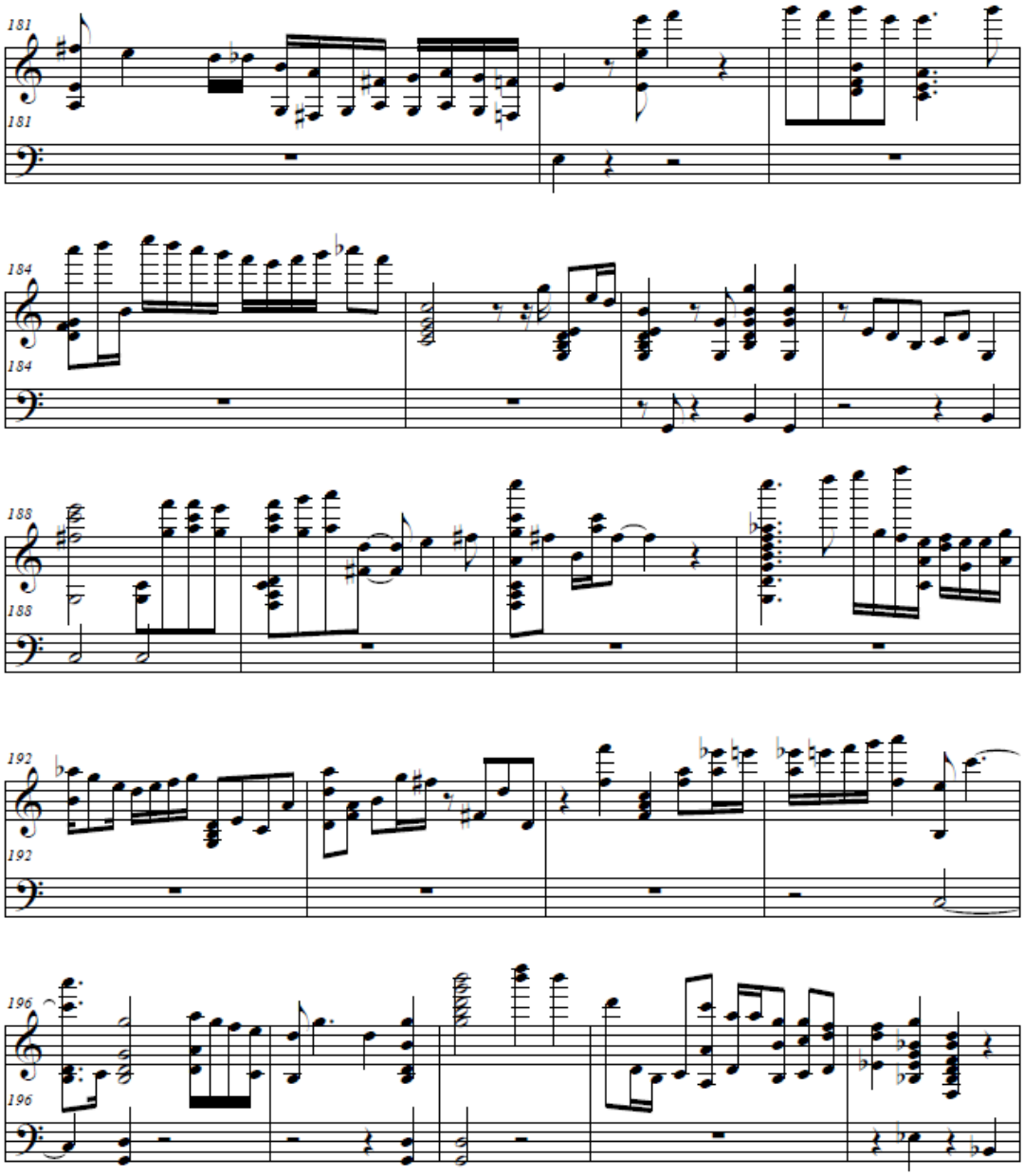

92
92
97
97
102
102
106
106
110
110

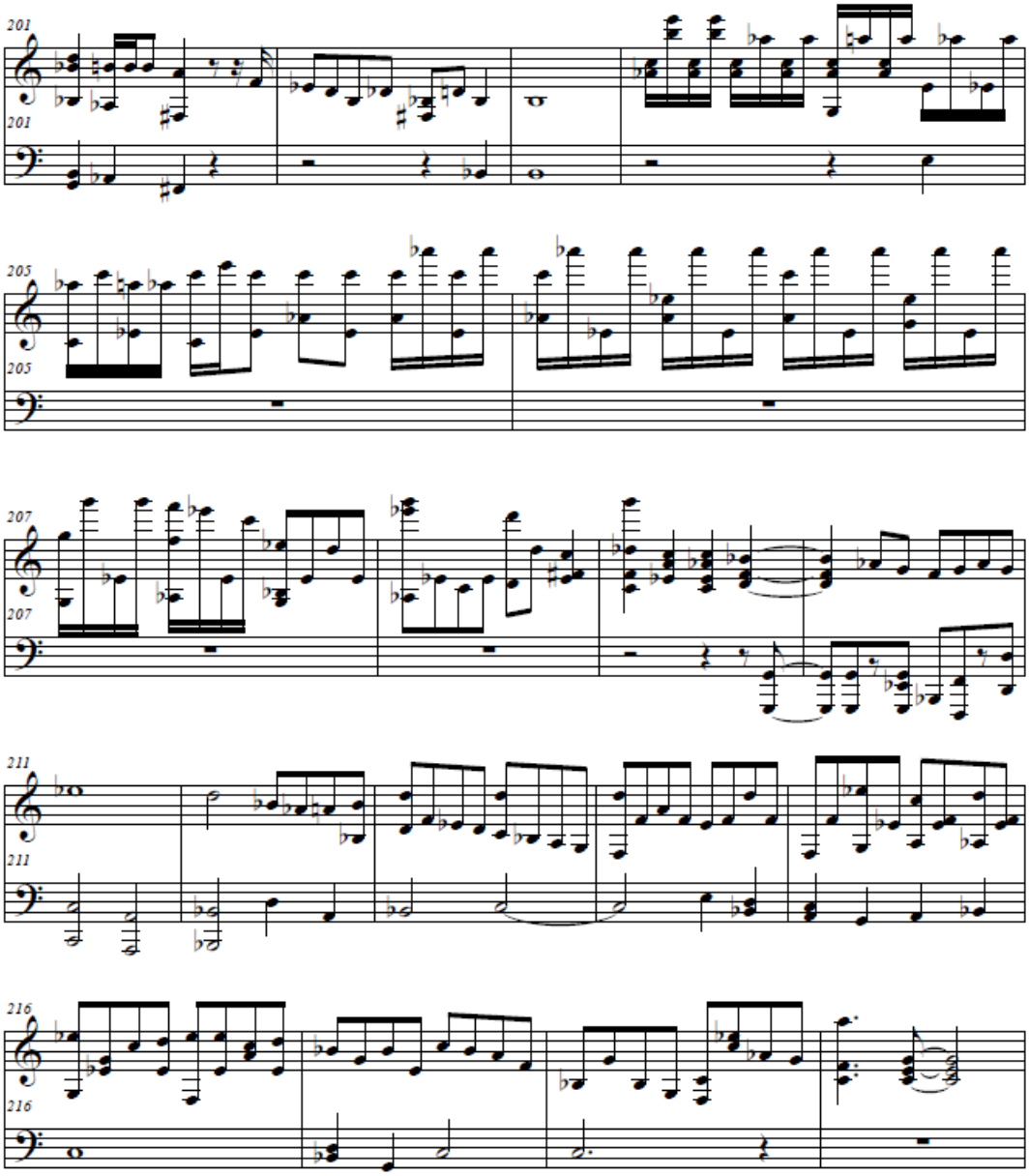

113
113
115
115
117
117
119
119
121
121

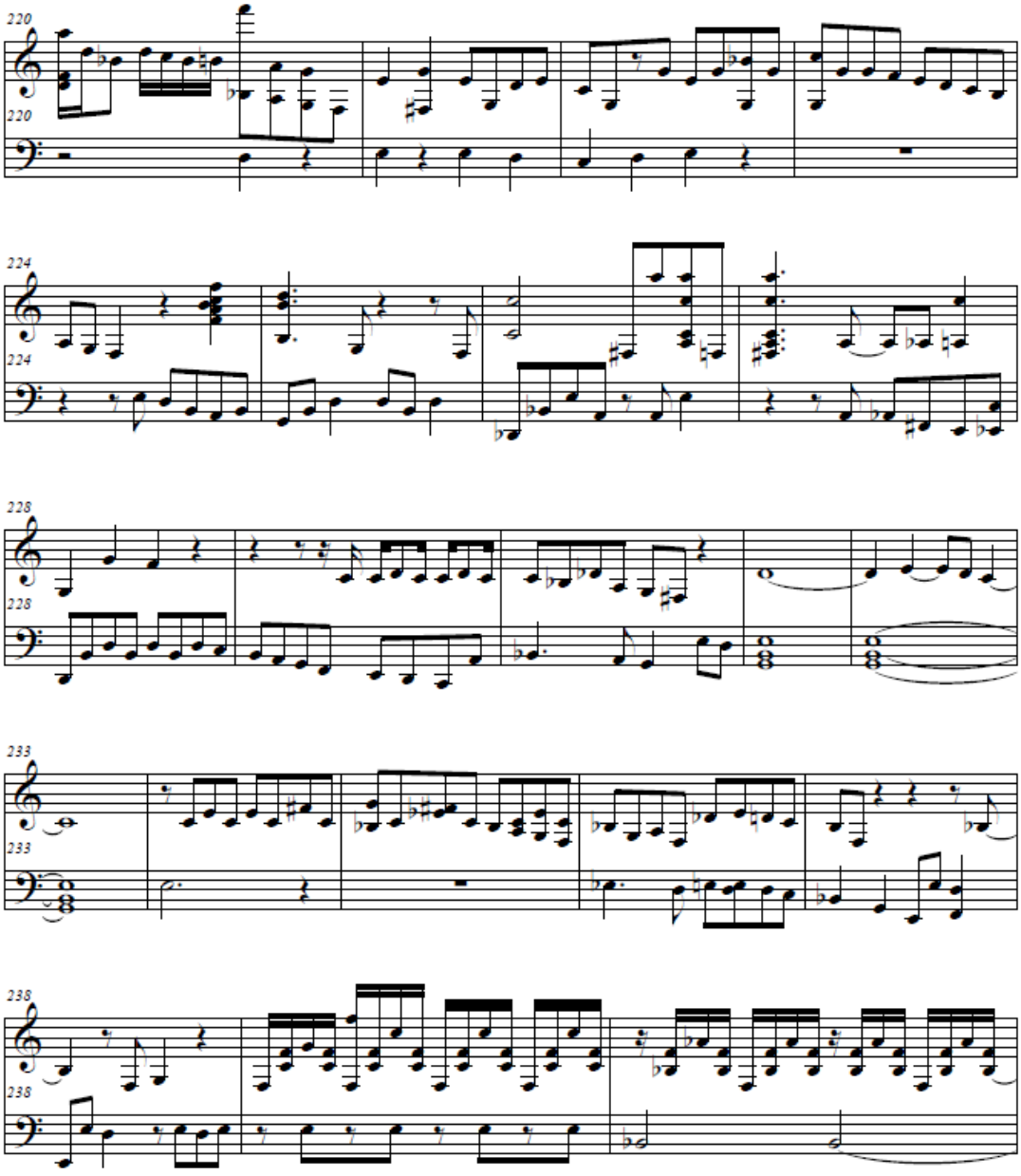

125
125
127
127
129
129
132
132
137
137

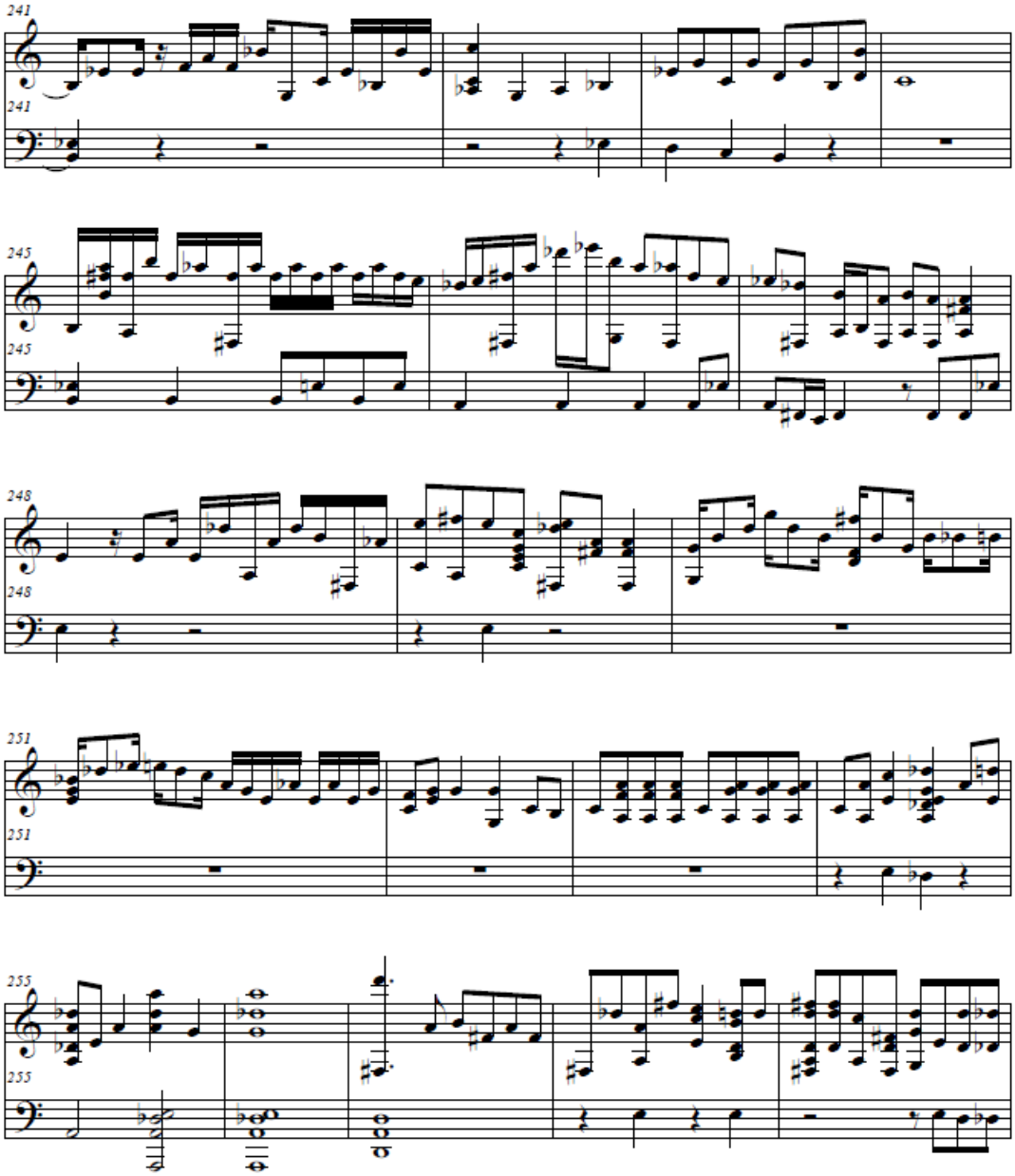

141
141
143
143
147
147
151
151
156
156

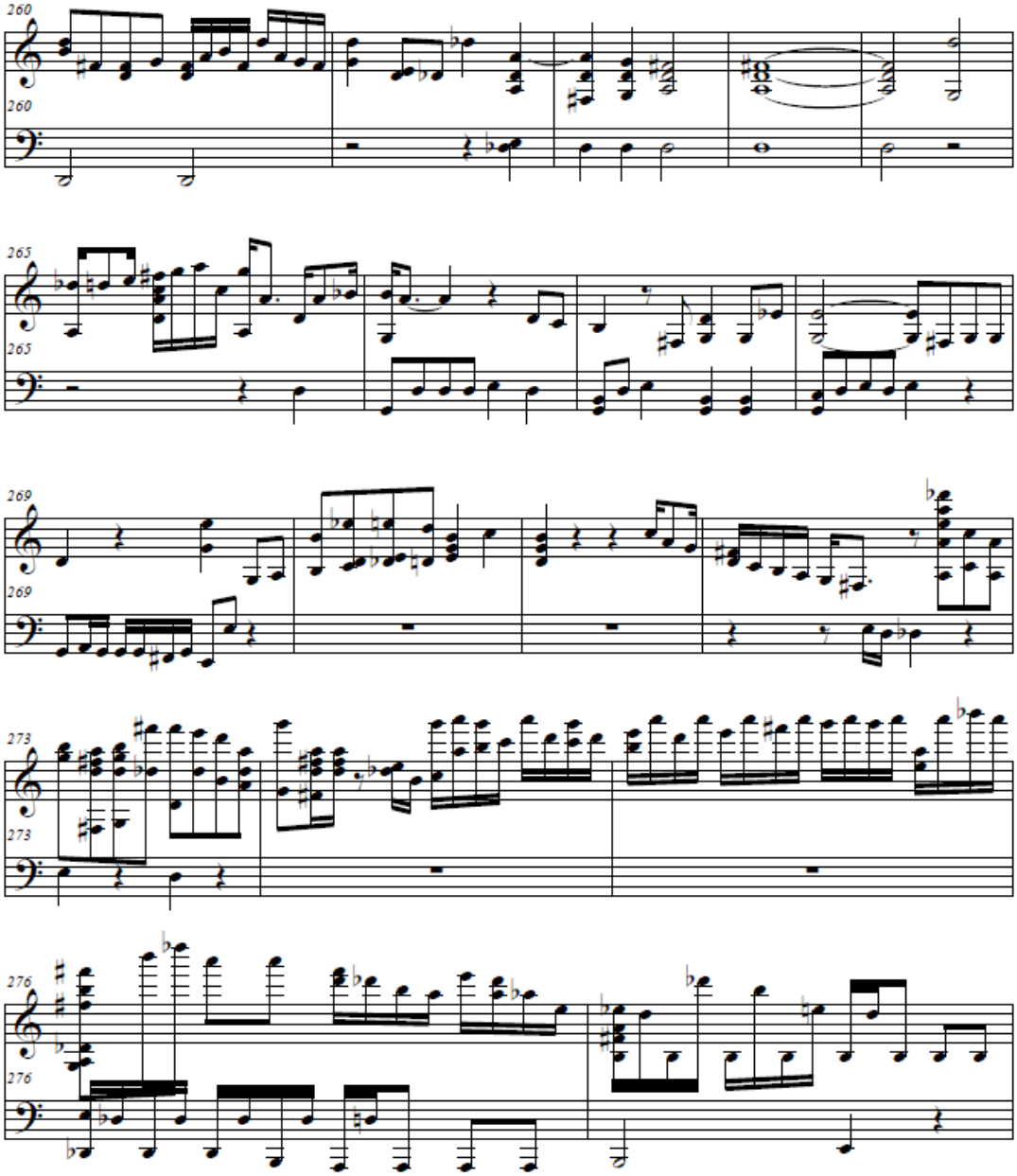

161
166
170
173
177

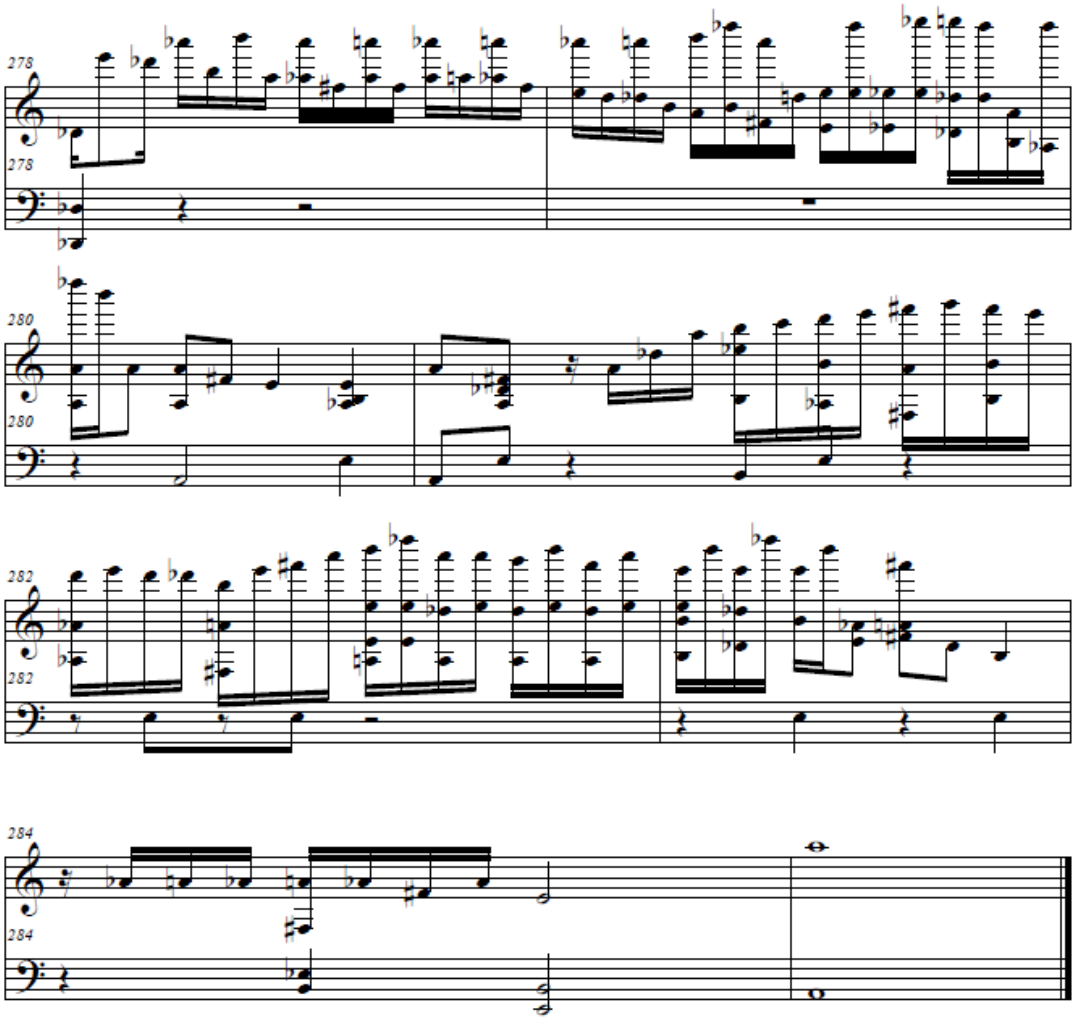

181
181
184
184
188
188
192
192
196
196


201
201
205
205
207
207
211
211
216
216


220
220
224
224
228
228
233
233
238
238


241
241
245
245
248
248
251
251
255
255


260
260
265
265
269
269
273
273
276
276


278
278
280
280
282
282
284
284