# Aprendizaje Automático

Segundo Cuatrimestre de 2021

Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

## Trabajo Práctico

Generación de Audio - WaveNet



| Integrante | LU | Correo electrónico |
|---|---|---|
| Jonathan Scherman | 152/15 | jonischerman@gmail.com |
| Axel Martín Savizky | 803/15 | axel.savizky@gmail.com |
| Facundo Linari | 591/16 | facundo.linari@hotmail.com |
| Luciano Strika | 76/16 | lucianostrika44@gmail.com |
| Joaquin Romera | 183/16 | joaquin.romera@gmail.com |



**Music WaveNet Generator Audio**

# Índice

# 1. Introducción

En este informe mostraremos el uso de deep learning para generar melodías musicales automáticamente. A través de *deep learning* y otras tecnologías que iremos explicando en el informe, entrenaremos modelos autorregresivos basados en convoluciones causales dilatadas, los evaluaremos y compararemos para poder concluir pro y contras de cada uno.

## 1.1. Breve repaso historico de la generación automática de música

Si uno ve a la música como un conjunto de notas, entonces la forma más fácil (aunque quizás poco efectiva) sería elegir cada nota aleatoriamente. Esto fue lo que hizo Mozart en 1787. Propuso generar canciones eligiendo notas aleatoriamente tirando un dado. Luego *evolucionó* esta forma tirando 2 dados y eligiendo la suma de ellos dos. Esto le permitió crear cerca de 272 tonos diferentes.

Más tarde, en 1950, Iannis Xenakis propuso pensar a la música usando estrictamente los conceptos de probabilidad y estadística. Esto fue llamado *Stochastic Music*[1]. La idea era definir a la musica como una secuencia de elementos (o sonidos) que ocurren con una determinada chance. Entonces, puede elegir elementos aleatoriamente basándose en conceptos puramente matemáticos, en particular, probabilísticos. Esta forma de ver la musica junto con la evolución del aprendizaje automático hace que estos dos temas puedan relacionarse y así permitir la generación automática de música.

## 1.2. MIDIs

'Describimos a la música como una serie de eventos sonoros organizados de una manera determinada. Estos eventos pueden ser representados de diversas maneras; históricamente se perfeccionó la escritura musical mediante una notación simbólica en pentagramas, pero el soporte digital por excelencia es el protocolo MIDI, desarrollado a principios de 1980 para facilitar el uso entre instrumentos de diferentes fabricantes. El protocolo define mensajes que permiten identificar, entre otros eventos, las notas que son ejecutadas, el momento en que deben ejecutarse y durante cuánto tiempo.

Necesitamos una representación simbólica de piezas musicales para poder abstraernos de aspectos accidentales: son las notas ejecutadas en cierto orden lo que definen una pieza, no la ejecución en sí. Por otro lado, la simplicidad del formato MIDI hace que computacionalmente se requiera muchísimo menos trabajo que al trabajar con audio ya que son menos las notas/tokens que si se usara una discretización directa de la onda de sonido.

## 1.3. Motivación

Poder generar música automáticamente tiene varios beneficios y motivaciones. Entre otros, podemos enumerar los siguientes:

- El estudio de la interrelación entre aprendizaje automático y la generación de música podría permitir a compositores encontrar inspiración y asistirlos a desarrollar ideas.
- Desde un punto de vista comercial inclusive deberíamos poder generar música programáticamente, sin tener que incurrir en gastos de derechos de autor, por ejemplo.
- Puede ser de interés la interacción entre tecnología y arte, y como una influye en la otra.

[1] https://www.sweetwater.com/insync/stochastic-music/

# 2. Desarrollo

Para poder entrenar nuestros modelos utilizamos un dataset con una colección de aproximadamente 70.000 canciones en formato MIDI. En esta colección encontramos una gran diversidad de géneros y estilos musicales, cada canción a su vez con una cantidad variable de instrumentos e información adicional. Decidimos procesar previamente el dataset para obtener un subconjunto más homogéneo que sea útil para el problema que intentamos resolver.

El procesamiento que hicimos se basó en utilizar únicamente las partes de cada canción correspondientes a instrumentos monofónicos tonales. Estos instrumentos nos aseguran reducir la complejidad del dataset ya que solo producen notas individuales, y es justamente la sucesión de notas en un orden determinado las que definen una melodía. A su vez, transportamos todas las canciones a la misma tonalidad para que compartan el mismo conjunto de notas posibles, reduciendo una vez más la información disponible en el dataset.

Estas tareas de manipulación de los datos fueron realizadas mediante Music21, una biblioteca de Python para el análisis y manipulación de datos relacionados con representaciones simbólicas de información musical.

Una vez procesados y filtrados los datos, nuestro siguiente objetivo será entrenar un modelo de maximum-likelihood autorregresivo para poder generar melodías con algún grado de coherencia musical o plausibilidad. Luego, los evaluaremos no solo cualitativamente (oyendo las melodías y juzgandolas) sino que usaremos como métrica la perplejidad sobre datasets de test.

Para explicar el modelo que buscamos entrenar, primero es necesario entender qué significa que sea autoregresivo. Una modelo autoregresivo, llevandolo al caso de la música, significa que para predecir una nota, se basará en todas las notas anteriores.

Por otro lado, dijimos que nuestros modelos usan las convoluciones causales dilatadas. Las convoluciones causales son un tipo de convolución[2] que se aseguran que el orden mediante el cual se modelan los datos, se mantiene. Esto es algo muy importante para los modelos autoregresivos.
Una convolución dilatada es una convolución donde el filtro se aplica sobre un área mayor que su longitud omitiendo los valores de entrada con un cierto paso. Es equivalente a una convolución con un filtro más grande derivado del filtro original dilatándolo con ceros, pero es significativamente más eficiente.

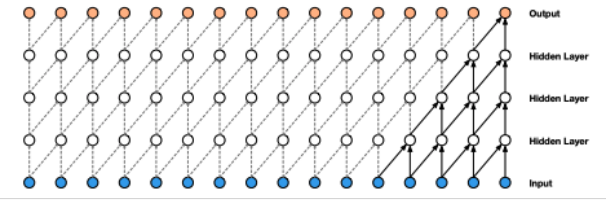


Figura 1: Capas de las convoluciones causales

[2]https://paperswithcode.com/method/convolution

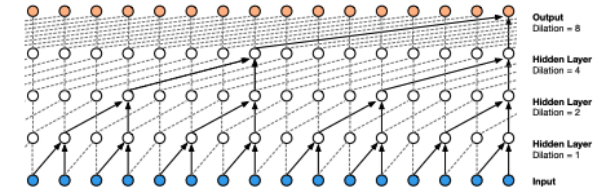


Figura 2: Capas de las convoluciones causales dilatadas

Otra parte que desarrollamos fue un método de evaluación. Necesitamos algún método objetivo de evaluación dado que solo escuchar las melodías generadas por el modelo y evaluarlas era algo subjetivo. Es por esto que elegimos evaluar a los modelos usando la perplejidad. La perplejidad está basada de alguna forma en "que tan sorprendido está un modelo que predijo las n notas anteriores, cuando se le pide la probabilidad de la nota n+1". Más formalmente, la perplejidad dado un modelo P y un conjunto de notas m, es la media aritmética de las inversas de $P(x_i|x_1, x_2, ..., x_{(i-1)})\forall x_i \in m$

Finalmente, para la generación de música utilizamos Beam Search. Como ya dijimos, nuestra red nos permite evaluar cual es la nota más probable dada una secuencia anterior. Queremos con esto poder generar melodías. Imaginemos un grafo en el que cada nodo representa una nota y que cada camino forma una melodía. Podríamos empezar desde la raíz eligiendo vecinos aleatoriamente para formar una melodía. Nosotros para utilizar la red que entrenamos usamos Beam Search. Beam search es un algoritmo greedy de búsqueda en un grafo. En nuestro caso consiste en, partiendo de un fragmento cualquiera, expandir con todas las posibles continuaciones del fragmento (lo cual es la frontera), de estos obtener los k mejores (los que la red indica que son más probables). Luego, solo se expandirán estos nodos seleccionados formando una nueva frontera para repetir el proceso. Esto se repite hasta formar una melodía del largo deseado. Un ejemplo esto (con k = 3) se puede ver en la figura 3. El k es un hyperparámetro con el cual experimentaremos.

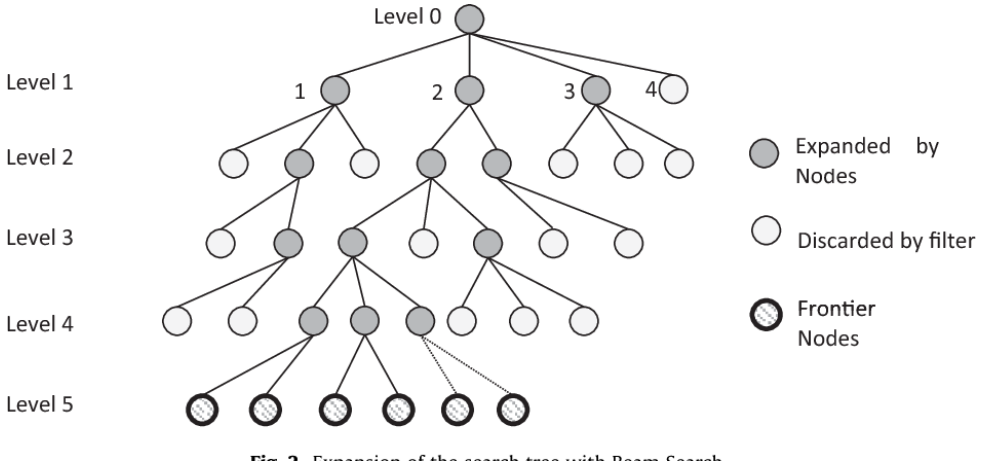


Figura 3: Expansión del árbol de búsqueda con Beam Search

Además se implementaron 2 heuristicas en la generación:

- Cantidad máxima de repeticiones: con un parámetro definir hasta cuantas veces se puede tocar la misma nota de forma seguida.
- Lista tabú: prohibir que se pueda tocar una de las j últimas notas tocadas. j se pasa como parámetro.

# 3. Resultados y Discusión

En esta sección mostraremos algunos modelos construidos y los compararemos. Algunos desafíos que enfrentamos para realizar la experimentación fueron los siguientes:

1. **Los tiempos de entrenamiento del modelo eran demasiado grandes** pues no contamos con herramientas suficientes para generarlos. El preprocesamiento de los datasets y la arquitectura propuesta requieren mucha fuerza de procesamiento y esto fue precisamente un problema. Para solucionarlo, tuvimos que, entre otras cosas, realizar optimizaciones del script usado para generar el modelo y reducir considerablemente el set de entramiento.

2. **No conocíamos métricas para comparar modelos que involucren generación de audio**. Investigando encontramos que una métrica frecuente utilizada en este área es la de *perplejidad*, que será detallada a continuación.

## 3.1. Métrica de perplejidad

## 3.2. Experimentación

Para construír el modelo, realizamos un script en Python en el cual, en primer lugar, cargamos en memoria una cantidad $k$ de canciones MIDI en memoria para luego entrenar el modelo utilizando el framework de redes neuronales *Keras*

Decidimos considerar distintos modelos variando la cantidad de canciones utilizadas como set de entrenamiento para lo cual entrenamos modelos de 100, 500, 100, 5000, y 10000 canciones. A su vez, nos pareció interesante analizar si había alguna diferencia con utilizar distintas funciones de activación en las neuronas de la red. Por lo tanto generamos, para cada modelo generado, 3 versiones usando las funciones de activación *sigmoid*, *tanh*(tangencial hiperbólica) y *relu*. La *figura 4* muestra el gráfico de estas funciones.

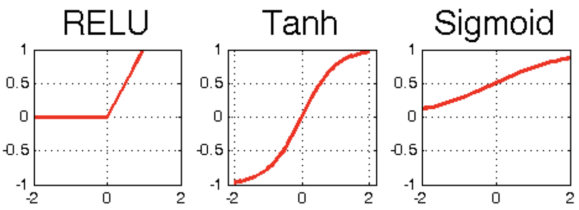


Figura 4: Funciones de activación utilizadas

Luego, los modelos generados terminaron siendo los siguientes:

- *Modelo 1*: #training = 100, activation_function = relu
- *Modelo 2*: #training = 100, activation_function = tanh
- *Modelo 3*: #training = 100, activation_function = sigmoid
- *Modelo 4*: #training = 1000, activation_function = relu
- *Modelo 5*: #training = 1000, activation_function = tanh
- *Modelo 6*: #training = 1000, activation_function = sigmoid
- *Modelo 7*: #training = 5000, activation_function = relu
- *Modelo 8*: #training = 5000, activation_function = tanh
- *Modelo 9*: #training = 5000, activation_function = sigmoid

- *Modelo 10*: #training = 10000, activation_function = relu
- *Modelo 11*: #training = 10000, activation_function = tanh
- *Modelo 12*: #training = 10000, activation_function = sigmoid

Para cada modelo listado, calculamos la perplejidad de 100 canciones y tomamos el promedio. Los resultados se muestran en la *figura 5*:

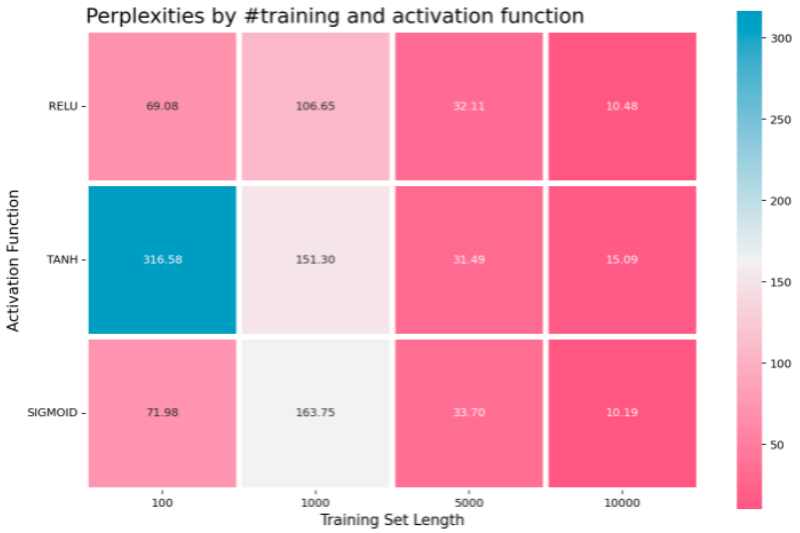


Figura 5: Perplejidades de modelo por funcion de activación utilizada y tamaño de set de entrenamiento. Cada valor representa al promedio de perplejidades aplicada sobre una muestra de canciones de test.

La *figura 5* parece indicar que, incrementando el tamaño del set de entrenamiento, obtenemos mejores modelos (usando como métrica la perplejidad obtenida, en este caso). Es curioso que esto no parece ser siempre así, puesto que se puede ver que *sigmoid-100* resulta mejor que *sigmoid-1000*. Ahora bien, podemos ver que este resultado parece valer cuando se consideran sets de training lo suficientemente grandes. Por último, los resultados muestran que la función de activación *relu* resulta en general la opción que en promedio otorga mejores resultados.

## 3.3. Generación de melodías

Para todos los siguientes extractos que se van a mostrar se utilizó como entrada Rammstein - Halleluja para comparar de la mejor forma a los distintos modelos. No había cambios significativos con otras canciones de entrada que se probaron.

Variando k de beam search no parece haber cambios significativos en las melodías generadas.

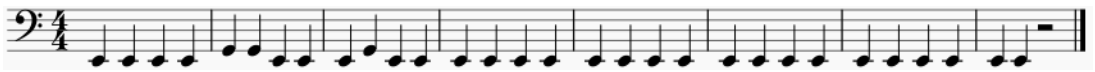

Figura 6: Melodía generada con modelo entrenado con 100 canciones, función de activación tanh y k=1

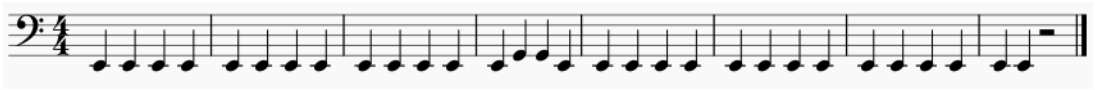

Figura 7: Melodía generada con modelo entrenado con 100 canciones, función de activación tanh y k=100

Variando cantidad de datos de entrenamiento. A mayor cantidad de datos de entrenamiento parece tender a variar menos de nota.

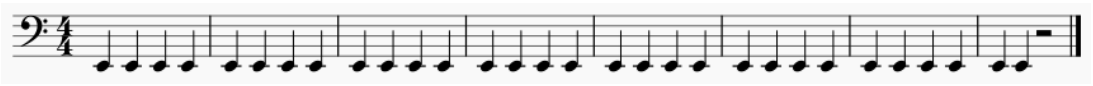

Figura 8: Melodía generada con modelo entrenado con 10000 canciones, función de activación tanh y k=1

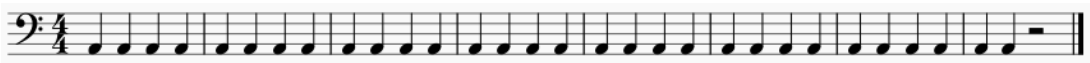

Figura 9: Melodía generada con modelo entrenado con 10000 canciones, función de activación tanh y k=100

Variando función de activación no hay cambios significativos. Sigue variando entre a lo sumo 3 notas. Cabe destacar que en la figura 10 las 3 notas que suenan forman al acorde La mayor, indicando cierto aprendizaje.

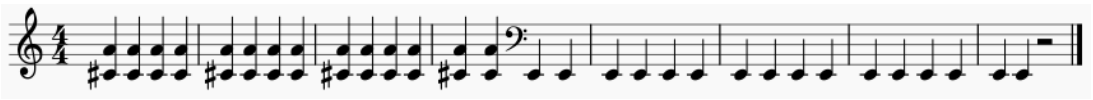

Figura 10: Melodía generada con modelo entrenado con 10000 canciones, función de activación relu y k=100

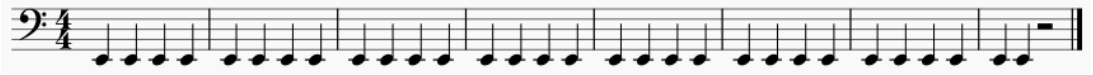

Figura 11: Melodía generada con modelo entrenado con 10000 canciones, función de activación sigmoid y k=100

Con máxima repetición de nota puede apreciarse que se sale de tocar siempre lo mismo, especialmente al subir el k.

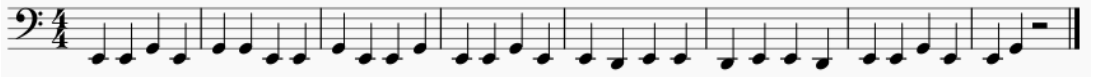

Figura 12: Melodía generada con modelo entrenado con 100 canciones, función de activación relu, k=1 y repetición máxima de nota de 2

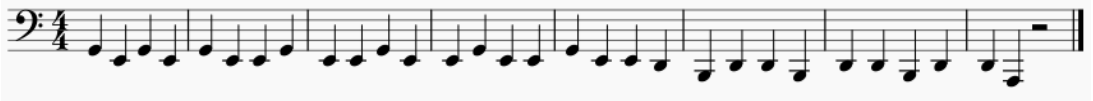

Figura 13: Melodía generada con modelo entrenado con 100 canciones, función de activación relu, k=100 y repetición máxima de nota de 2

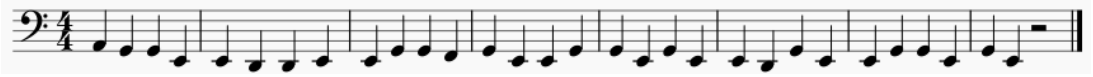

Figura 14: Melodía generada con modelo entrenado con 10000 canciones, función de activación tanh, k=100 y repetición máxima de nota de 2

Con lista tabú se puede apreciar que a mayor k mejora la melodía formada.

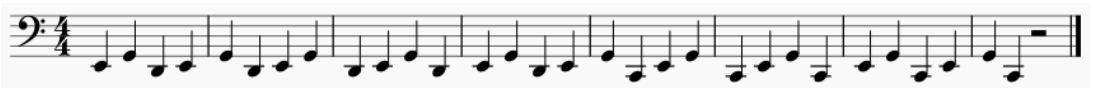

Figura 15: Melodía generada con modelo entrenado con 100 canciones, función de activación relu, k=1 y lista tabú de tamaño 2

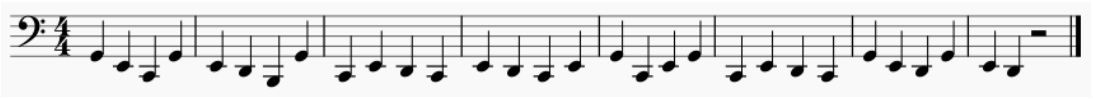

Figura 16: Melodía generada con modelo entrenado con 100 canciones, función de activación relu, k=100 y lista tabú de tamaño 2

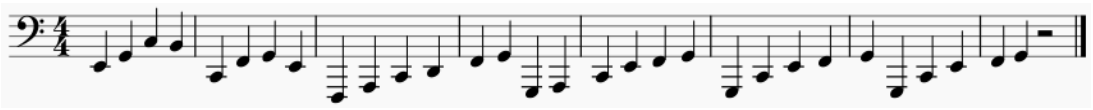

Figura 17: Melodía generada con modelo entrenado con 10000 canciones, función de activación sigmoid, k=1000 y lista tabú de tamaño 4

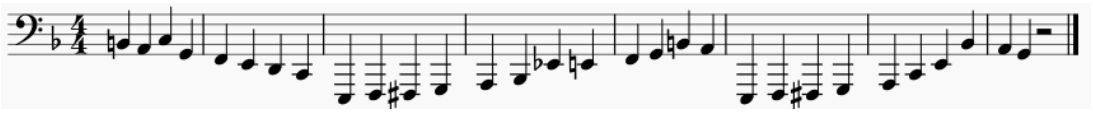

Figura 18: Melodía generada con modelo entrenado con 10000 canciones, función de activación sigmoid, k=1000 y lista tabú de tamaño 8

# 4. Conclusiones

En este trabajo nos planteamos implementar un generador de melodías MIDI utilizando la técnica de *WaveNet*, mediante redes neuronales convolucionales.

Pudimos realizar una experimentación en la cual generamos distintos modelos y los comparamos, utilizando las métricas descritas como perplejidad, obteniendo como resultado que, de los modelos generados, el mejor parece ser el que tiene el input set más grande (10k) y usa *relu* como función de activación en las neuronas.

Los resultados del análisis cualitativo no fueron del todo positivos. En los primeros modelos se obtuvieron "melodías"que constaban de apenas un par de notas distintas repetidas. A pesar de esta deficiencia, cabe destacar que el modelo .ªprendió.ªlgo de la jerarquía tonal inherente a las canciones con las que se lo entrenó, obteniendo intervalos consonantes, la cual es fundamental para la música tonal occidental.

Como trabajo futuro, queda pendiente trabajar con muestras más grandes (que en este trabajo no pudimos hacer por falta de recursos) y más acordes al problema que intentamos solucionar: será ideal contar con un dataset conformado únicamente por melodías, sin ninguna información adicional. También hacer un pre-procesamiento más exhaustivo de la muestra en el que podamos incluir no solo las notas, sino además las duraciones de las mismas y los silencios entre ellas, e incluso usar training sets específicos para obtener resultados determinados desde algún punto de vista estético (como, por ejemplo, entrenar al modelo utilizando canciones de un género en particular y ver que genere resultados en los que puedan destacarse características fundamentales de dicho género).

# Referencias

[1] *WAVENET: A GENERATIVE MODEL FOR RAW AUDIO*: https://arxiv.org/pdf/1609.03499.pdf

[2] *Visual Transformers*: https://arxiv.org/abs/2010.11929

[3] *Gwern*: https://www.gwern.net/GPT-2-music