Universidad Zaragoza
1542

Trabajo Fin de Grado

Título del trabajo:
Síntesis de instrumentos musicales realistas mediante redes neuronales

English tittle:
Musical instruments synthetised through neural networks

Autor/es

Sergio Bosque Torrente

Director/es

José Ramón Beltrán Blazquez

Escuela de Ingeniería y Arquitectura

2016-2017

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. Sergio Bosque Torrente,

con nº de DNI 18067413Q 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, (Título del Trabajo) Síntesis de instrumentos musicales realistas mediante redes neuronales,

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

Zaragoza, 24 / Noviembre / 2017

Fdo: Sergio Bosque Torrente

Resumen

Este proyecto se encuentra enmarcado en el campo del procesado digital de audio, que se encarga de aspectos relacionados con la digitalización, edición, síntesis y reproducción de sonido.

El proyecto se centra en la búsqueda de un modelo de red neuronal que permita realizar la síntesis de música con instrumentos más realistas que los que es capaz de generar ahora mismo el sintetizador MIDI. El modelo elegido será adaptado para poder condicionarse con archivos de este tipo.

Dado que la síntesis musical mediante redes neuronales es un campo muy reciente de investigación nos centraremos en evaluar los modelos disponibles en el momento de la presentación del proyecto, dando valor añadido a aquellos que han demostrado ser capaces de dar los resultados deseados y, debido al propósito del proyecto de dar un método no dependiente del aparato que actúe como reproductor, se valorará muy especialmente la carga computacional del modelo.

Evaluaremos los dos modelos que han sido capaces de dar resultados satisfactorios y haremos una breve revisión del proceso que ha llevado llegar a ellos.

El primero del que hablaremos será WaveNet, un modelo de red neuronal convolucional (CNN, Convolutional Neural Network) en una dimensión desarrollado por el equipo DeepMind de Google para la síntesis de voz realista en distintos idiomas a partir de texto. Este modelo está basado en dos trabajos anteriores del mismo equipo, PixelRNN y PixelCNN, una red neuronal recurrente (RNN, Recurrent Neural Network) y una CNN respectivamente, diseñadas para reconstruir imágenes a partir de un fragmento de la misma. Este modelo en sí ha sido perfeccionado posteriormente por Magenta para formar Nsynth, un modelo compuesto por dos redes con el formato de WaveNet que permite sintetizar notas de instrumentos monofónicos de duración variable, permitiendo cosas tan interesantes como mezclar dos instrumentos ya existentes para dar lugar a uno nuevo con características de ambos.

El segundo del que hablaremos será SampleRNN, un modelo RNN que genera audio muestra a muestra, como WaveNet pero con un peso computacional significativamente menor. Es mucho más ligero por varias razones, la más importante es que es un modelo con varias capas de RNN apiladas que se ejecutan a distintas velocidades, por lo que no todas las capas deben ejecutarse para cada muestra y además, el modelo necesita una cantidad menor de tiempo y muestras para entrenarse. Como punto extra, este modelo ha sido capaz de obtener sin condicionamiento algo parecido a música, tanto de instrumentos individuales como de orquesta, mientras que WaveNet es más apropiado para instrumentos individuales monofónicos y aun con éstos tiene ciertas dificultades.

Índice

1. Introducción

1.1. Objetivos, alcance y contexto del proyecto

El objetivo principal del proyecto es ofrecer una alternativa a los medios actuales de reproducción de música, principalmente para el formato MIDI. Si este formato tuviese un medio de reproducción universal podría ser usado como el modelo de transmisión de música instrumental con mayor compresión existente y con las ampliaciones que ha ido recibiendo a lo largo de los años, más las que podrían llegar, se podría convertir en un formato de transmisión de eventos útil en cualquier situación.

El problema que presenta MIDI actualmente para reproducir audio es que el resultado es completamente dependiente del aparato que lo esté reproduciendo. El sintetizador que reproduzca el archivo puede estar implementado de múltiples formas, y todo depende de cómo lo quiera hacer el fabricante. Las principales formas en que están incluidos hoy en día es en forma de sintetizadores aditivos o sustractivos, que se sustentan en grandes cantidades de osciladores, bancos de filtros, unidades de efectos… o en forma de concatenadores de sonidos pregrabados (de forma similar a como se usan para la síntesis de voz), o mediante otro tipo de implementaciones que son puro software pero que tienen el defecto de causar ciertos retrasos no deseados.

Obviamente, no vamos a ofrecer un modelo perfectamente funcional en el momento de la entrega de este proyecto, puesto que es un campo que se ha empezado a explorar muy recientemente (el documento que especifica los detalles de WaveNet [9] fue publicado el 19 de septiembre de 2016 y es el modelo más antiguo que ha sido capaz de dar resultados). Sin embargo, los resultados hablan por sí solos, literalmente, Google ha perfeccionado WaveNet [8] para poder usarlo en su nuevo asistente personal, dándole una voz que podría pasar por la de un ser humano (de momento sólo en inglés y en japonés) y para que todos sus equipos sean capaces de ofrecer esta funcionalidad la síntesis de la voz, que es extremadamente exigente en términos de computación, la realiza de forma distribuida en sus servidores. Por otro lado, el modelo del que parte SampleRNN [12], aunque estaba inicialmente pensado para generar texto, está dando resultados superiores a los inicialmente mostrados por WaveNet y, sin alcanzar todavía los resultados que actualmente ofrece, está ganando reconocimiento rápidamente gracias a que es muchísimo menos pesado en términos de computación de lo que es WaveNet básico y por ello requiere mucho menos tiempo para entrenarlo y para poder generar resultados aun con menos medios disponibles.

Para concluir los objetivos, no buscamos que sea capaz de memorizar exactamente la estructura a pequeña ni a gran escala, sino que sea capaz de generalizar a partir de los datos que le ofrecemos para poder ofrecer al oyente música real a partir de archivos de peso mínimo.

1.2. Organización y planificación del proyecto

Tal y como se encuentra en la memoria, el proyecto se ha explorado de los aspectos más generales a los más específicos, empezando por las herramientas a utilizar, que serían python, tensorflow y pytorch, continuando por comprender la estructura de los modelos a utilizar y el concepto de red neuronal como tal y terminando por una exploración profunda de ambos modelos y su comparación.

Durante el proceso de desarrollo del proyecto realizamos varias reuniones para poder poner en común las nuevas alternativas que habíamos podido encontrar que pudiesen resultar útiles. Algunas de las alternativas más interesantes se nombrarán posteriormente y se verá por qué han sido descartadas y cuál es la razón que las hace alternativas válidas.

1.3. Introducción breve de las herramientas

1.3.1. Python

Se trata de un lenguaje de programación interpretado cuya principal filosofía es hacer que el código sea más legible. Como lenguaje no es excesivamente eficiente a la hora de trabajar con números, pero se extiende gracias a la gran cantidad de módulos desarrollados por los usuarios para aumentar sus funcionalidades base, además permite interactuar fácilmente con otros lenguajes, lo que favorece a módulos como numpy, con una base de C que le permite operar con números y estructuras de datos con gran eficiencia, solucionar las posibles pegas del lenguaje. Es de los lenguajes más utilizados a la hora de desarrollar redes neuronales, favorecido por las diferentes librerías diseñadas para ello que se le han añadido.

1.3.2. Tensorflow

Es una biblioteca de código abierto liberada por Google para el desarrollo de programas de aprendizaje automático, entre ellos pero no exclusivamente redes neuronales. Permite desarrollar grafos fijos de las redes con las que se va a trabajar de forma previa al entrenamiento para poder luego optimizar cómo se van a realizar las operaciones sobre ellos. Admite ejecución sobre GPU mediante CUDA sin necesidad de cambiar el código y es especialmente bueno con computación distribuida.

En el proceso de creación de redes neuronales Tensorflow [6] trabaja algo más a bajo nivel, por lo que resulta en un código algo difícil de leer pero por suerte, como ya hace unos cuantos años que fue liberado se han creado otros módulos que permiten trabajar a más alto nivel partiendo de Tensorflow, aunque algunos tienen algún problema de rendimiento.

1.3.3. PyTorch

Es otra librería de código abierto para python diseñada para aprendizaje automático. A diferencia de tensorflow, pytorch respeta algo más la esencia propia de python, siendo mucho más orientada a objetos complejos y permitiendo desarrollar grafos que pueden cambiar en tiempo de ejecución sin afectar al rendimiento. Permite también ejecutar sobre GPU mediante CUDA simplemente trasladando los tensores a la GPU. Todavía se encuentra en beta pero los resultados son prometedores. [7]

1.4. Introducción a las redes neuronales

1.4.1. ¿Qué es una red neuronal?

Una red neuronal artificial (ANN, Artificial Neural Network) [5] es un sistema de computación inspirado por la forma en que trabajan las neuronas cerebrales. Son habitualmente referidas como aproximadores universales de funciones. Lo más importante que poseen es su capacidad de aprendizaje.

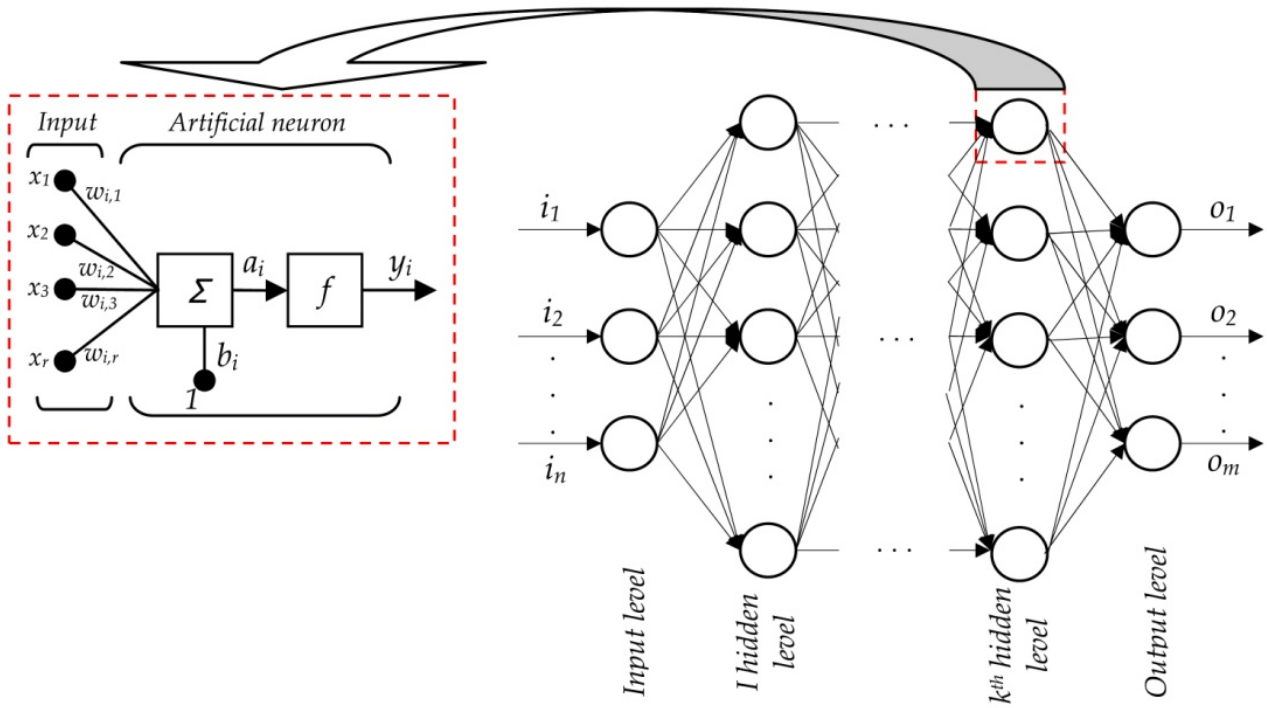


Figura 01. Esquema básico de una red neuronal

Como puede observarse en la figura, cada neurona de estos sistemas consta de varias partes, un input donde recibe datos que pueden provenir de otras neuronas o de un input de datos creados por el usuario a los que se les aplica unos pesos correspondientes, una función de transferencia (representada aquí como un sumatorio) a la que se le puede aplicar un bias, que es un valor que habitualmente se usa por si las entradas son cero que la red siga siendo capaz de dar un resultado coherente, y una función de activación que da lugar a la salida. Entre estas funciones de activación algunas de las más habituales son RELU (máximo entre cero y la entrada a la función), la función sigmoide (que devuelve valores entre cero y uno con pequeñas variaciones para cambios en grandes valores o muy pequeños) y la tangente hiperbólica (devuelve valores entre menos uno y uno y se usa principalmente cuando es necesario usar valores negativos).

En la parte derecha de la figura se puede observar la estructura típica de una red neuronal, donde la entrada pasa a través de varias capas ocultas hasta llegar a la capa que devuelve la salida. Las capas intermedias no tienen por qué ser todas iguales, ni siquiera tienen que estar totalmente conectadas con la anterior ni únicamente con la capa anterior.

1.4.2. Redes Neuronales Convolucionales

Las CNN [4, 5] son redes basadas en el funcionamiento de la corteza visual humana. En ellas hay una serie de bancos de filtros que realizan convoluciones en dos dimensiones que permiten extraer características propias de la imagen con la que estén tratando. Tras los diferentes bancos de filtros es habitual reducir las dimensiones de las imágenes que devuelven mediante unas capas de agrupación que reducen las dimensiones de lo que es devuelto por estos bancos de filtros. Tras atravesar todo este proceso es habitual que en sistemas de clasificación haya una o varias capas completamente conectadas a la anterior para permitir diferenciar qué hay en la imagen. Es algo distinto para redes generadoras de este tipo.

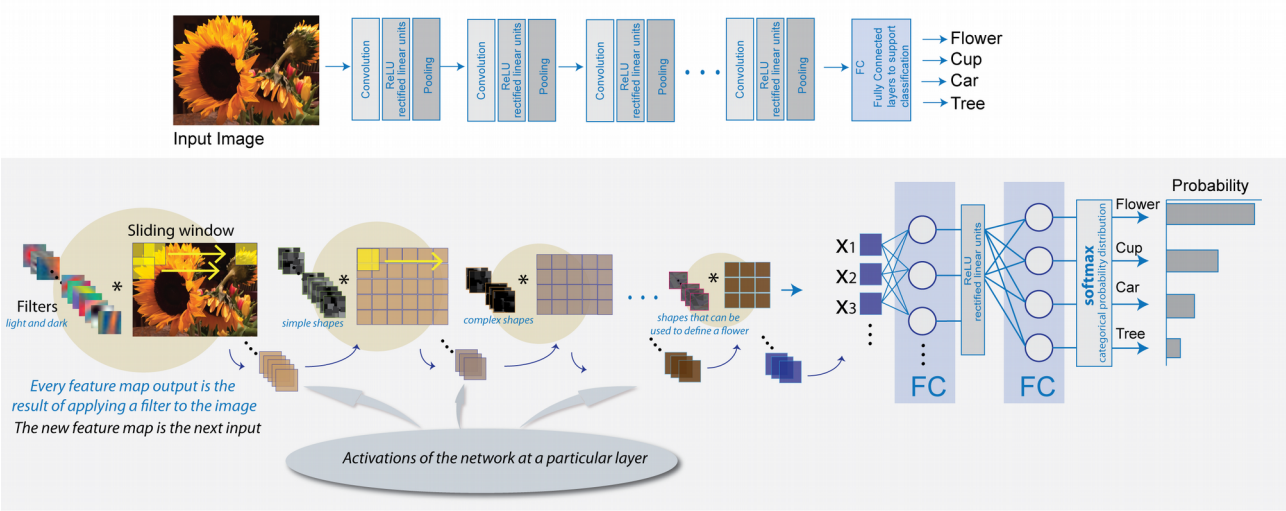


Figura 02. Esquema de una red neuronal convolucional

Dentro de este tipo de redes hay una estructura muy popular para imagen que se consideró como una posible alternativa, ésta sería llamada Generative Adversarial Networks o GAN. Es un sistema que se basa en dos redes, un convolucional y una deconvolucional, una se encarga de generar lo que es solicitado y la otra distingue si lo que ha generado es correcto, haciendo que se mejoren mutuamente y permitiendo crear sistemas muy prácticos con aprendizaje no supervisado.

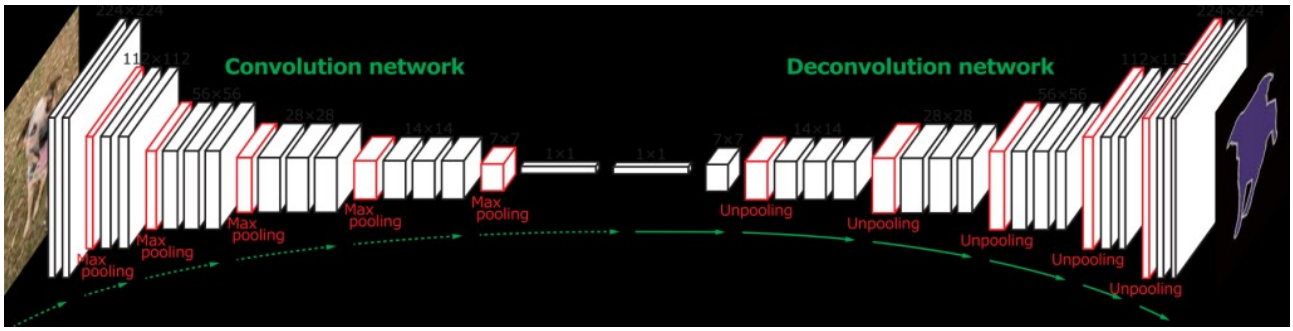
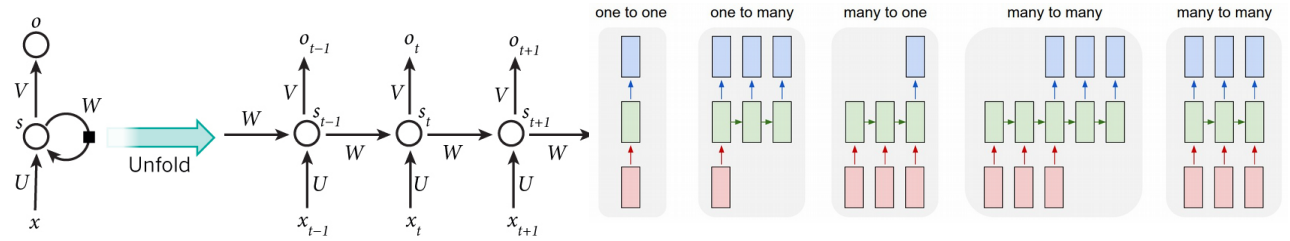


Figura 03. Estructura de una red neuronal tipo GAN

Este sistema fue finalmente descartado por la dificultad de implementación, puesto que para poder trabajar con MIDI era necesario diseñar una red que admitiese un número variable de entradas y pudiese dar lugar a un número variable de salidas.

1.4.3. Redes Neuronales Recurrentes

Son aquellas donde la salida de una neurona de la misma capa se usa como entrada de la siguiente y la salida de la red suele usarse posteriormente como entrada. [4, 5]



Figuras 04 y 05. Estructura de una red neuronal recurrente y formatos en que se suelen usar

Como puede observarse pueden tener muchos tipos de estructuras diferentes.

Uno de los problemas habituales que suelen presentar este tipo de redes es el sobrecondicionamiento, esto es que aprenden demasiado de los datos de entrada y se vuelven incapaces de generalizar, lo que causa que tan sólo sean capaces de reproducir los datos que se les han introducido aunque con múltiples permutaciones. Las soluciones que se usan para arreglar este problema son usar algunos tipos de neuronas más complejas que las habituales para poder “olvidar” parte de lo que han aprendido y, pese a que esto supone aumentar un poco el coste computacional, suelen ser redes bastante ligeras y de rápida ejecución. Los modelos de neurona con capacidad de olvido más utilizados son LSTM (Long Sort Term Memory) y GRU (Gated Recurrent Unit), siendo la diferencia entre ambas que las neuronas GRU olvidan de forma completamente automática, mientras que en las LSTM se puede elegir qué es lo que van a olvidar. Las neuronas GRU poseen cierta ventaja a la hora de ser procesadas ya que internamente llevan a cabo menos operaciones, es por ello que el proyecto está desarrollado sobre este tipo de neuronas.

2. Estado del arte

2.1. Exploración de los modelos a utilizar

2.1.1. WaveNet

Es un modelo basado en CNN, concretamente en PixelCNN, para la generación de audio muestra a muestra. A diferencia de muchos de los modelos de CNN no dispone de capas de agrupación y va reduciendo la dimensionalidad con la que trabaja mediante convoluciones dilatadas [1, 8].

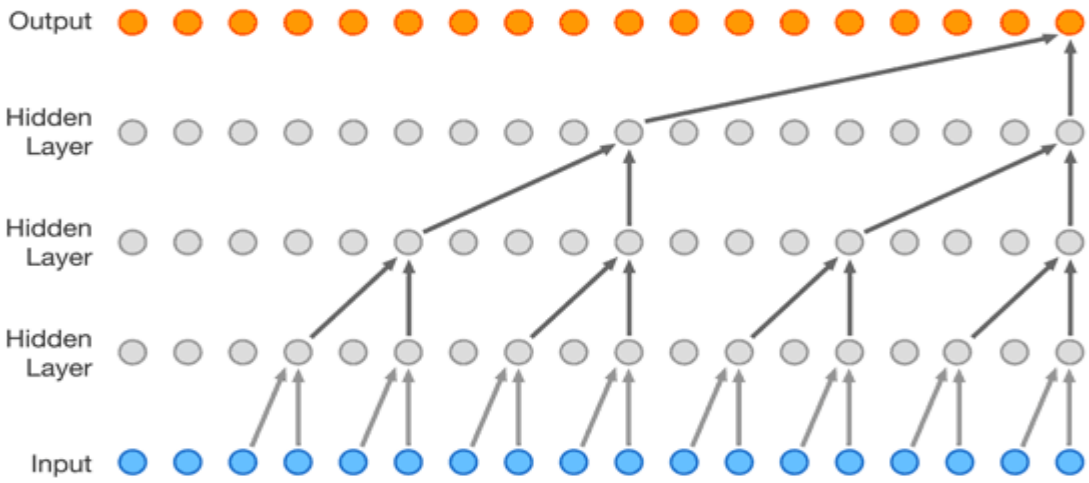


Figura 06. Estructura de WaveNet

Cada neurona realiza convoluciones de dos muestras de la capa anterior, y conforme se va escalando en la estructura se saltan más neuronas de computar hasta que se llega a la salida, donde sólo se computa una. Este hecho hace el modelo algo más escalable, sin embargo, conforme más alcance se quiere que tenga el modelo el número de capas y de neuronas escala exponencialmente. Algunas de las mejoras que se introdujeron para aliviar algo este hecho fue recordar parte de las muestras que se han calculado previamente para reducir el tiempo de generación.

Acumulando los diferentes tipos de mejoras que admite el modelo base, se puede llegar a optimizar hasta el punto de, ejecutándolo sobre la CPU, generar a mayor velocidad que a tiempo real [16, 18]. Entre estas mejoras se incluye: reducir el número de neuronas distintas por capa a uno, lo que reduce la flexibilidad mucho pero permite reutilizar el máximo posible las muestras generadas, eliminar la no linearidad de la función de activación por una aproximación lineal, lo que reduce extremadamente el tiempo de generación, reimplementar el modelo en un lenguaje fuertemente tipado que permita tener un mayor control del uso de memoria y recursos como sería C y por último optimizar la forma en que se genera trabajo para los hilos que se procesan de acuerdo al tipo de CPU que se está utilizando [16].

El modelo ha demostrado ser capaz por si sólo de mejorar los mejores resultados obtenidos en la síntesis de habla mediante aplicaciones TTS (Text To Speech) en las que anteriormente se usaban modelos paramétricos o concatenativos. La mejora ha sido sustancial, tanto en la calidad del audio como en la inteligilibilidad del mismo. Alimentando el modelo con muestras de piano sin condicionamiento ha demostrado ser capaz de crear audio de piano bastante convincente, sin ser capaz de mantener una estructura consistente a largo plazo.

Posteriormente otro equipo de Google, Magenta, quienes se dedican a extender modelos de redes neuronales con un propósito artístico, crearon Nsynth partiendo de este modelo [11].

Nsynth es un modelo compuesto por dos redes del tipo de WaveNet, que funcionan en cierta forma como lo hacen las anteriormente nombradas GAN. La primera de las redes que la compone obtiene unas características autoaprendidas de los datos de entrada y se los transmite a la segunda red, quien se encarga de generar el audio a partir de las características reducidas en tiempo que le da la primera.

El modelo da unos resultados ciertamente interesantes, puesto que permite crear sonidos de instrumentos que no existen y poseen sonidos propios de varios instrumentos reales. Entre las demos que han liberado se encuentran dos que permiten mezclar entre dos y cuatro instrumentos distintos.

Pese a todas las mejoras que han hecho sobre el modelo, resulta extremadamente pesado en términos de entrenamiento, tanto en tiempo como en cantidad de datos, costándole casi dos semanas entrenarse al modelo extractor de características y casi un mes de entrenamiento al modelo generador ser capaces de dar resultados, y eso contando con una gran cantidad de GPUs trabajando en paralelo. Si alguien decidiese mejorar los resultados obtenidos, debería partir de este modelo para poder usar un modelo ejecutado sobre ambos, CPU y GPU, para obtener un modelo capaz de generar en tiempo real. La siguiente imagen muestra cómo sería el modelo optimizado [16]:

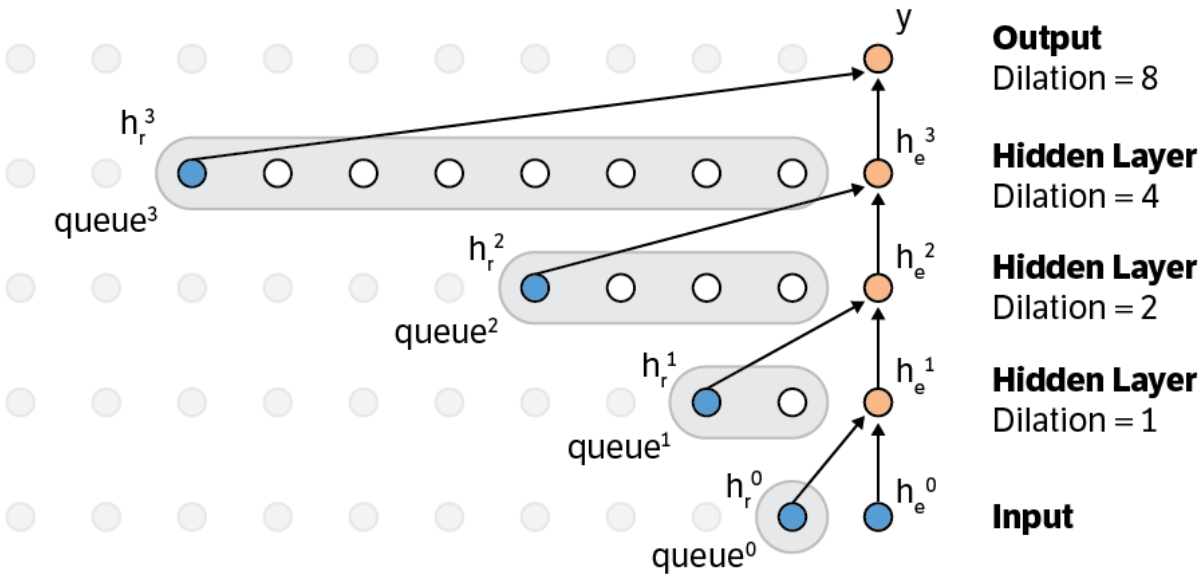


Figura 07. WaveNet tras aplicar optimización

Además, parece ser que hay algún detalle omitido en el documento donde especifican su funcionamiento sobre cómo realizar el condicionamiento local [10] para poder generar muestras condicionadas por los archivos MIDI, puesto que nadie ajeno a Google ha sido capaz de obtener los resultados mostrados por DeepMind en su artículo [8].

2.1.2. SampleRNN

Definida como “una red fin-a-fin no condicionada para la generación de audio” por sus autores, esta red está basada en capas apiladas de RNN que funcionan a distintas velocidades [1, 12, 13]. Cada nivel de los que forman la red se ejecuta una fracción de las veces que se ejecuta el nivel inmediatamente inferior y le ofrece un condicionamiento de las muestras que esté procesando a cada nodo que depende de él directamente. Se acumulan varios de estos niveles hasta que el último se encarga de procesar directamente a nivel de muestra y dar el output.

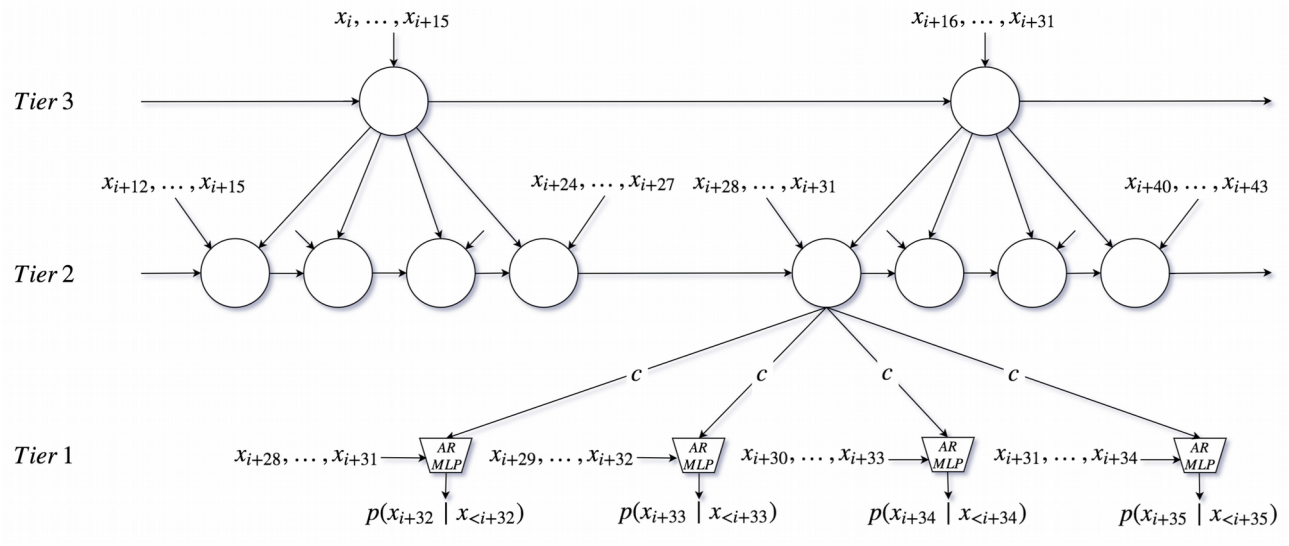


Figura 08. Estructura de SampleRNN desenrollada

En la imagen se puede observar la estructura que tendría la red desenrollada si se usan tres capas y en cada capa se reduce a la cuarta parte el número de veces que será procesada respecto a la inferior.

La idea procede del experimento de Karpathy para crear un modelo RNN capaz de procesar texto conservando cierta estructura a largo plazo. Al igual que SampleRNN utilizaba varias capas apiladas de RNN [14, 17] pero con distinto alcance y funciones de activación de las que usa SampleRNN.

2.2. Comparación de ambos modelos

Ambos modelos son bastante pesados en términos de computación, así que les cuesta tiempo tanto entrenarse como generar muestras, sin embargo para ser capaces de ver exactamente cuánta diferencia hay entre ambos modelos es adecuado valorar su tamaño y la cantidad de operaciones que debe realizar cada una tanto para generación como para entrenamiento.

Empecemos valorando WaveNet. Los modelos que han sido capaces de dar resultados en este campo tienen habitualmente 30 ó 40 capas en el modelo generador (extraído del código de Nsynth) [11]. En tiempo de generación esto supone que con ejecutar dos multiplicaciones, una suma y una vez la no linearidad [5] por capa es suficiente como para poder generar muestras. En tiempo de entrenamiento sin embargo, cada vez que se actualizan los pesos y los bias es necesario volver a generar las muestras pasadas de nuevo, lo que lo vuelve a hacer un modelo extremadamente pesado y hace que tome mucho más tiempo entrenarlo.

En el caso de SampleRNN en tiempo de generación no se ejecuta de forma continua todo el modelo [13], permitiendo ganar bastante tiempo con respecto al tiempo de entrenamiento. Al igual que con WaveNet, en tiempo de entrenamiento esta ventaja no es posible aplicarla, puesto que todas las capas se están evaluando para obtener la siguiente muestra y poder actualizar los pesos y bias de todas las capas. Esto hace que el modelo sea bastante pesado de nuevo, pero, a diferencia de WaveNet, por la estructura interna basada en RNN y su comportamiento, no necesita tantos datos ni tiempo para ser capaz de dar resultados satisfactorios [2, 4, 5].

A la hora de tratar con el condicionamiento la complejidad extra de WaveNet también hace que sea algo difícil realizar modificaciones y ver su efectividad, mientras que SampleRNN es bastante más personalizable. Es posible condicionar correctamente ambos, pero debido a la diferencia de tiempo en el entrenamiento que existe entre ambos es más fácil evaluar distintos tipos de condicionamiento sobre SampleRNN.

Para finalizar las comparaciones cabe destacar que mientras que WaveNet sólo es capaz de entender instrumentos monofónicos correctamente, hecho razonable teniendo en cuenta que fue diseñado para generar la voz de una sola persona, SampleRNN es capaz de crear audio con cierta coherencia a largo plazo de múltiples instrumentos sonando a la vez aun sin condicionamiento.

3. Generación de la base de datos

3.1. Características de la base de datos para las redes neuronales

Para poder obtener resultados en un tiempo razonable, dado el tipo de equipos disponibles, las características de los archivos de audio no podían ser demasiado exigentes. Básicamente los archivos debían ser en formato .wav, con una frecuencia de muestreo de 16kHz y valores de amplitud de 16 bits, que luego manipulamos aplicándoles la ley mu para reducirlos a 8 bits con las menores pérdidas posibles.

Además de los archivos de audio, es necesario un archivo MIDI que se corresponda con cada uno. La correspondencia se ha usado en el modelo a partir del nombre.

La duración en tiempo de ambos archivos debe ser estrictamente inferior a cinco segundos para permitir que durante el entrenamiento la red no memorice de más las secuencias que se le están presentando y se vuelva incapaz de generalizar. Esto es debido a que la mayoría de secuencias suelen ser cíclicas, da igual el género del cuál sea la canción de la que procede el fragmento MIDI.

La base de datos completa no podía abarcar el espectro completo de combinaciones entre notas e instrumentos presentes en MIDI debido a que el tiempo que le cuesta converger al modelo es bastante alto. Para garantizar este hecho, hemos recopilado una colección de MIDI perteneciente al mismo autor.

3.2. Obtención de la base de datos

3.2.1. MIDI

Las piezas originales seleccionadas son 258 archivos con duraciones comprendidas entre los 45 segundos y 11 minutos, siendo la gran mayoría de una duración entre tres y cuatro minutos. Los archivos en el formato que la red necesita se crean haciendo recortes de los MIDI originales con un nombre en el siguiente formato AA-D-CCC-PPP, donde AA representa el álbum, D representa el disco, CCC es la canción dentro de ese disco y PPP representa la parte de esa canción.

La división en partes se hace teniendo en cuenta que todas las notas incluidas tienen tanto el evento NoteON como el evento NoteOFF en ese espacio de tiempo, a excepción de la percusión donde sólo hay eventos NoteON. Además de esto, se preprocesa el archivo MIDI para que los tics que indican los puntos donde inician los eventos estén en milisegundos para que sean más sencillos de adaptar a la red posteriormente.

3.2.2. WAV

Los WAV se han obtenido a partir de los fragmentos MIDI ya obtenidos. La idea inicial era usar muestras de instrumentos reales que se correspondiesen con el MIDI que se le daba a la red, pero debido a la forma que era necesario recortarlos para que la red funcionase correctamente no fue

posible. Los archivos se exportaron del MIDI con el mismo nombre y las características descritas.

3.3. Estructura de la base de datos

Todos los archivos se encuentran en un directorio raíz por debajo de donde se ejecuta el modelo. Si consideramos la dirección donde se encuentra el modelo como “/”, los archivos se encuentran bajo la dirección “/db”, estando los MIDI en la ruta “/db/MIDI” y los WAV en la ruta “/db/WAV”. Dado que tienen los mismos nombres los distintos archivos, para acceder a ellos desde el modelo se recopila la lista de archivos de uno de los dos directorios y después se buscan ambos y se preprocesan para adaptarlos a las necesidades de la red.

Los ficheros de test son tanto las canciones completas como algunas de otros autores pero cuyos instrumentos se encuentran incluidos entre los que se ha entrenado.

4. Estructura de la red

4.1. Modelo seleccionado

Dado que lo más importante en este proyecto es ser capaz de generar muestras a partir de un modelo condicionado por archivos MIDI y precisamente lo que necesitamos evaluar es el condicionamiento, esto requiere realizar bastantes pruebas durante el entrenamiento, por lo que hemos decidido emplear SampleRNN [14].

Ahora explicaremos la estructura concreta que posee la red (sin incluir el condicionamiento a partir de MIDI, que se encuentra en el apartado siguiente). La estructura está dividida en tres capas, siendo dos obligatorias y pudiendo generar capas superiores en función de la profundidad que se le quiera dar al modelo. Todos los parámetros se configuran cuando la red es generada.

La capa inferior es ejecutada cada muestra y se trata de un perceptrón multicapa que se encarga de devolver la muestra siguiente como una probabilidad para cada una de las 256 posibilidades. Esta capa procesa cada vez que se ejecuta 16 muestras e incluye el condicionamiento generado por la capa inmediatamente superior.

La capa intermedia y la capa superior poseen una estructura interna semejante y la diferencia entre ellas es la cantidad de muestras que procesan y cada cuánto se ejecutan. Ambas son una RNN de dos capas con una capa de expansión que adapta la salida para condicionar a la capa inmediatamente inferior. Ambas procesan una cierta cantidad de muestras y el condicionamiento de la capa inmediatamente superior, la capa superior al no tener una por encima no procesa condicionamiento [15], y cuando llega de nuevo a esa cantidad de muestras que procesa las evalúa otra vez. La capa intermedia procesa 16 muestras, al igual que la inferior, pero se ejecuta una vez cada 16 muestras y la capa superior hace lo mismo pero con 64 muestras.

Este tipo de estructura permite a la red memorizar patrones a distintas escalas, otorgándole memoria a largo plazo de las formas de onda que han sucedido. El mero hecho de usar RNN otorga al modelo una cantidad de memoria sorprendente y el hecho de apilar varias capas con RNN permite que cada una se especialice en memorizar patrones en una escala distinta. Por ello a este modelo se le pueden continuar añadiendo capas superiores sin un coste excesivo, con el correspondiente riesgo de perder capacidad de generalizar claro está.

En este caso el condicionamiento se está ejecutando una vez por milisegundo, así que está condicionando a la capa intermedia, porque si se intentase condicionar la superior se perdería el beneficio de no tener que ejecutarla tan a menudo.

4.2. Estructura del condicionante

Partiendo del tipo de estructuras que suelen usarse para generar MIDI con un audio coherente para un solo instrumento, los que mejores resultados daban eran aquellos que generaban nuevas notas basándose en cuáles habían sido las anteriores, no sólo teniendo en cuenta el tiempo entre ellas sino también cuáles se estaban tocando a la vez y tratando todas las notas con el mismo tipo de procesado. Dado que en MIDI el orden de las notas no implica necesariamente que éstas sean equidistantes en términos de distancia frecuencial es necesario tratar cada nota de forma individual. Algo similar puede aplicarse a los instrumentos soportados por este formato, que estén en el orden que se encuentran no quiere decir necesariamente que tengan alguna relación directa en sus timbres. Es por esto que la mejor opción que valoramos fue usar una matriz de pequeñas redes RNN individuales para cada nota y cada instrumento, que no tuvieran que ser evaluadas si no hay notas de su correspondiente par nota-instrumento.

Esta estructura permite garantizar el mismo tratamiento independientemente de la nota y el instrumento que se trate [15], además de permitir realizar la misma nota del mismo instrumento simplemente duplicando una pequeña parte del modelo y que tenga coherencia temporal en cada nota. Para transformar los datos que da esta capa hemos utilizado una capa convolucional de expansión que permite extraer datos en una dimensión.

Para llegar a esta estructura las pruebas que realizamos fueron las siguientes: el primer intento de condicionante fue una pequeña red de tipo WaveNet [8] a la que le pasábamos como input vectores en los que se le indicaban qué instrumento estaba tocando, la nota, el tiempo restante y la velocidad de pulsado indicada en MIDI (que se corresponde en cierta forma con el volumen al que tiene que sonar), este modelo no era capaz de generar un condicionamiento que permitiese a la red generalizar, además de tener la pega de ser bastante pesado y no permitir más de un par nota-instrumento que se estuviese ejecutando a la vez.

El segundo intento fue una red RNN de tres capas (no tres capas apiladas de RNN como en SampleRNN, sino una sola red RNN), se le entregaba lo mismo que a la anterior, pero era más ligera y el condicionamiento que generaba se podía aprovechar hasta cierto punto para que la red generalizase. Dado que sufre de los mismos problemas a la hora de ejecutar instrumentos que el anterior intento decidimos hacer que procesase un número fijo de vectores, pero esto llevaba a que si ese número era excedido en algún momento durante el archivo MIDI parte de la información se perdía y si era con muy pocos instrumentos-notas tocando a la vez se malgastaba tiempo de computación.

El tercer intento fue una máquina de Boltzmann restringida [3]. Tenía resultados similares al segundo intento y sufría de las mismas pegas al tratar con archivos MIDI, pero el modelo era mucho más ligero a cambio de perder cierta coherencia a medio plazo en el condicionamiento generado, lo que hacía que las notas largas no llegasen a durar lo que debían en los archivos generados.

El último intento fue el que nos solucionó todos los problemas que estábamos teniendo [15]. El modelo permitía que las combinaciones de nota-instrumento que no estuviesen presentes en el archivo se desconectaran y no fuesen afectadas por el ajuste de pesos y bias, lo cual permitía ajustar

el coste computacional del modelo en función de las notas presentes en el archivo y entrenar cada note e instrumento de forma individual. Lo único que necesita cada elemento de la matriz como entrada es la velocidad indicada en MIDI. Dado que la percusión no necesita tener eventos de NoteOFF ni de velocidad cero (son comandos equivalentes) es el único caso donde el valor de la velocidad se le pasa sólo cuando aparece el evento NoteON, para el resto de instrumentos el valor de la velocidad se mantiene hasta que recibe un evento NoteOFF. Este modelo permitiría crear múltiples instancias de cada par nota-instrumento, lo que permitiría incluso sintetizar múltiples instrumentos tocando la misma nota pero es algo que en este trabajo no hemos implementado.

4.3. Experimentos realizados

Inicialmente evaluamos WaveNet [8] sin condicionamiento, para obtener una estimación del tiempo que necesitaba para converger mientras buscábamos más información del estado del arte en generación muestra a muestra de audio. En tres semanas no fue capaz de converger, puede que no estuviese completamente bien implementado el modelo de partida que usamos, pero durante ese tiempo encontramos información sobre las redes GAN, así que decidimos buscar a ver si existía alguna red de ese tipo diseñada para trabajar en una sola dimensión. Dado que este tipo de red es empleado principalmente para generar imágenes a partir de otros datos no fuimos capaces de encontrar nada que nos permitiese aproximarnos a el resultado que pretendíamos obtener.

Durante el periodo de búsqueda de GAN nos topamos por casualidad con el modelo del que parte SampleRNN [12], un modelo similar en estructura pero que procesaba texto en lugar de audio y nos llamó la atención. Resultaba muy interesante tratar de investigar sistemas que fuesen buenos procesando texto porque los modelos que habíamos encontrado hasta ese momento habían sido empleados en versiones reducidas para procesar texto y obtenían buenos resultados, así que era posible que un sistema que permitiese procesar texto fuese capaz de procesar audio de la forma que necesitábamos. Así fue de hecho.

SampleRNN era capaz de converger en aproximadamente una semana y era capaz de obtener buenos resultados para la mayoría de instrumentos e incluso le daba igual que se le alimentase con audio de múltiples instrumentos (hecho que por lo que se ve en el post de DeepMind sobre WaveNet, no le agradaba al modelo). Una vez obtenido el modelo que íbamos a usar hicimos las pruebas de condicionamiento que se han nombrado antes.

5. Resultados

5.1. Comparativa con el estado del arte

Aunque los resultados que hemos obtenido no llegan siempre a alcanzar los del estado del arte actual, dada la falta de recursos con respecto a las grandes compañías y la pequeña base de datos que teníamos resulta razonable la diferencia. Aunque el modelo ahora mismo no es capaz de dar lo mejor de sí mismo, es el primer modelo condicionado por MIDI que es capaz de generar audio para múltiples instrumentos de forma simultánea. Nsynth [11] es el otro modelo actual que está condicionado por MIDI y permite generar audio, y, aunque permite hacer cosas muy interesantes como mezclar varios instrumentos o incluso sonidos reales entre si y tocar canciones con eso, no es capaz de generar más de una nota cada vez, lo que, en caso de tener múltiples notas simultáneas, supone que varias copias del modelo tendrían que estar corriendo en paralelo para obtener resultados y, teniendo en cuenta el coste computacional que tiene este modelo, no es algo viable para el ordenador de un usuario normal.

5.2. Análisis de los resultados

Al escuchar muestras generadas por el modelo resulta agradable. Suenan los instrumentos que deben y la forma de onda que se puede observar es similar a la que se genera al usar un sintetizador MIDI, con lo que podemos asegurar que es capaz de memorizar la estructura de la forma de onda a corto y largo plazo. El modelo genera una pequeña cantidad de ruido que no es perceptible mientras esté sonando algún instrumento, sin embargo es poco recomendable eliminar ese ruido con más entrenamiento porque el modelo dejará de ser capaz de generalizar los resultados, así que es un defecto aceptable. Este error es también nombrado en el documento original de la especificación de SampleRNN y también el por qué no hay que eliminarlo con más entrenamiento con más detalle.

Adaptando levemente la forma de tratar los eventos MIDI se podría crear un generador en tiempo real de audio que genere pequeñas redes condicionantes a pedido y optimizando un poco el modelo, o trasladando el condicionamiento a la versión más optimizada de WaveNet de la que se hacen varias referencias en este documento, se podría llegar a crear un generador universal que permitiese comprimir de forma extrema los archivos de audio, por lo que el resultado obtenido es altamente satisfactorio.

6. Conclusiones

Pese a que los resultados no son perfectos, son plenamente satisfactorios. La gran cantidad de aproximaciones e implementaciones disponibles para solucionar los tipos de problemas que se nos puedan ocurrir es inmensa. Pese a que la idea inicial era generar un modelo propio capaz de dar resultados a partir de MIDI, debido a la complejidad que supone crear un modelo desde cero que sea capaz de dar algún resultado, nos ha parecido mucho más interesante condicionar un modelo ya existente para que sea capaz de hacer lo que necesitamos. Hay un gran margen de mejora, pero ahora que hemos dado con un método para condicionar estas redes de una forma efectiva es mucho más claro como poder sacarle partido a ambos modelos, lo que, con una base de datos más grande para alimentar al modelo, permitiría entrenar a WaveNet sin problemas y ser capaz luego de reimplementarlo para que se pueda ejecutar en CPU [16] y GPU para poder sacarle el máximo rendimiento y generar a un tiempo superior al real, lo que permite quitar parte de las limitaciones impuestas en este modelo, como serían el aplicar la ley mu para reducir el tamaño de las muestras y reducir la frecuencia de muestreo a 16 kHz. Esto también permitiría que la red no use sólo MIDI precreado, sino que se pueda construir un sistema a su alrededor que permita alimentarlo de comandos MIDI en tiempo real, siendo realizados por músicos.

7.Desde dónde se puede continuar

Como ya he nombrado en las conclusiones, lo primero que habría que hacer sería trasladar el condicionamiento que hemos usado para SampleRNN a WaveNet, a un modelo que aproveche el guardado de muestras en buffers como ya he especificado para luego poder trasladarlo a otro lenguaje de forma óptima [16].

El segundo paso sería obtener una base de datos de MIDI con mayor variedad y, aprovechando el tratamiento que ya le doy a los archivos completos, adaptarlo para las necesidades de la red. En caso de no querer o no poder trasladar el condicionamiento a WaveNet este paso también sería recomendable, dado que los archivos que he empleado para trabajar con esta red no poseían suficiente variedad de instrumentos como para poder completar la matriz de condicionamiento. Existe un repositorio de Google que posee 3 Terabytes de archivos MIDI de múltiples géneros de música y diferentes instrumentos, así que podría ser un buen punto de partida.

El tercer paso sería reimplementar las no linearidades presentes en el modelo [19] con el que se esté trabajando de forma que sean una aproximación linear en el rango de valores estimado que va a trabajar el modelo, de manera que se reduciría el coste temporal de evaluarlas a aproximadamente la quinta parte (basándome en las pruebas que he hecho con la tangente hiperbólica) de lo que se beneficiarían tanto el entrenamiento como la generación si se ha estimado correctamente el rango de valores de trabajo.

Si se ha conseguido trasladar el condicionamiento correctamente a WaveNet [8], una vez que se ha realizado el entrenamiento sería recomendable trasladar los pesos y bias obtenidos de la parte correspondiente a WaveNet (sin incluir el condicionamiento) y trasladarlos a una implementación diseñada exclusivamente para generar muestras en la CPU como he nombrado antes. Dado que python permite comunicarse con módulos en otros lenguajes con facilidad no debería ser muy problemático el ejecutar parte del modelo en la CPU y parte en la GPU para sacarle máximo rendimiento.

8. Bibliografía / Referencias externas

8.1. Base de conocimientos

https://medium.com/@saxenauts/speech-synthesis-techniques-using-deep-neural-networks-38699e943861 [1]

http://slazebni.cs.illinois.edu/spring17/lec26_audio.pdf [2]

https://medium.com/@oktaybahceci/generate-music-with-tensorflow-midis-4bf928a35c3a (parte es útil para el condicionamiento) [3]

https://cloud.google.com/blog/big-data/2017/01/learn-tensorflow-and-deep-learning-without-a-phd (muy práctico para obtener una base en los tipos de redes que se encuentran en este trabajo y cómo se implementan en Tensorflow) [4]

http://www.deeplearningbook.org/ (útil para obtener base matemática y mejor comprensión de los modelos) [5]

https://www.tensorflow.org/ [6]

http://pytorch.org/ [7]

8.2. WaveNet / Nsynth

https://deepmind.com/blog/wavenet-generative-model-raw-audio/ (descripción original de WaveNet) [8]

https://arxiv.org/pdf/1609.03499.pdf (papel sobre WaveNet) [9]

https://github.com/ibab/tensorflow-wavenet (implementación de WaveNet, se puede elegir entre esta y la usada en Nsynth como punto de partida) [10]

https://magenta.tensorflow.org/nsynth (Nsynth de magenta, incluye enlaces al código) [11]

8.3. SampleRNN

https://arxiv.org/abs/1612.07837 (papel con la información sobre SampleRNN, aunque se encuentra algo incompleto en cuanto a cómo se está operando dentro del modelo) [12]

http://deepsound.io/samplernn_first.html (explicación más comprensible del modelo que en el enlace anterior aunque en menor profundidad, incluye enlaces al post original del modelo) [13]

http://deepsound.io/samplernn_pytorch.html (explicación de los errores del papel y enlace a la implementación en pytorch) [14]

8.4. Útil para entender el condicionamiento

http://www.hexahedria.com/2015/08/03/composing-music-with-recurrent-neural-networks/ (el punto de partida para el razonamiento tras el condicionamiento) [15]

8.5. Sobre la optimización

http://on-demand.gputechconf.com/gtc/2017/presentation/s7544-andrew-gibiansky-efficient-inference-for-wavenet.pdf (optimización para WaveNet) [16]

http://research.baidu.com/svail-tech-notes-optimizing-rnns-differentiable-graphs/ (optimización para RNN, SampleRNN ya cumple con la mayoría de estas características por su diseño pero se puede revisar para intentar mejorarlo) [17]

http://research.baidu.com/deep-voice-production-quality-text-speech-system-constructed-entirely-deep-neural-networks/ (en relación a la optimización de WaveNet) [18]

https://varietyofsound.wordpress.com/2011/02/14/efficient-tanh-computation-using-lamberts-continued-fraction/ (aproximación lineal de la tangente hiperbólica) [19]