Proyecto final

Composición de melodías en formato MIDI

Juan Carlos Guzmán Medina

Junio 2020

1 Introducción

Una red neuronal artificial es un modelo inspirado en la forma en que el cerebro humano procesa la información. Biológicamente, las unidades fundamentales del cerebro son las neuronas, las cuales reciben/envían señales entre ellas.

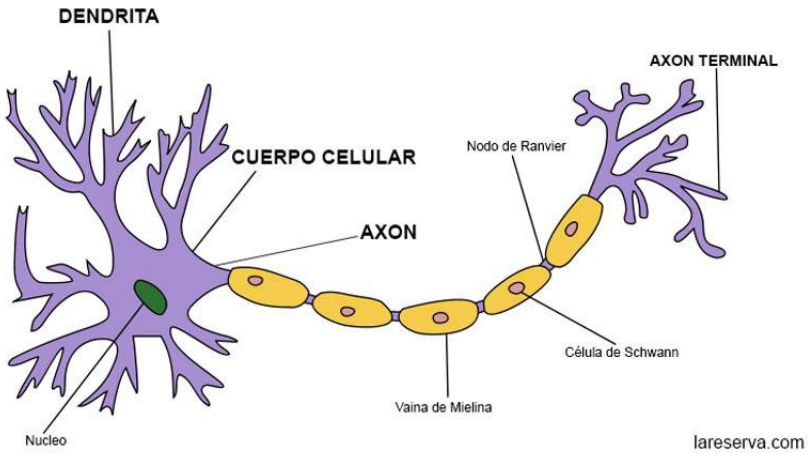


Imagen tomada de
https:
//medium.com/@williamkhepri/redes-neuronales-que-son-a64d022298e0

"El cerebro consiste en uno o varios billones de neuronas densamente interconectadas. El axón (salida) de la neurona se ramifica y está conectada a las dentritas (entradas) de otras neuronas a través de uniones llamas sinapsis. La eficacia de la sinapsis es modificable durante el proceso de aprendizaje de la red."[1].
Las redes neuronales artificiales tienen como unidad básica a cada unidad de procesamiento (neurona), donde cada una de ellas está interconectada con otras

y generalmente se organizan en capas, las cuales normalmente se dividen en 3 partes: capa de entrada, capa(s) oculta(s) y una capa de salida.
Las redes neuronales artificiales pueden clasificarse por su topología:

- Monocapa o perceptrón simple. Compuesta por una sola capa de entrada que envía la información a una capa de salida.
- Multicapa o perceptrón multicapa. Dispone de un conjunto de capas intermedias entre la capa de entrada y la de salida.
- Convolucional. Las neuronas de cada capa se unen solo con un subconjunto de neuronas de la capa siguiente, con lo que se logra reducir el número de neuronas necesarias, así como la complejidad computacional.
- Recurrente. No tiene estructura de capas, si no que permite realizar las conexiones entre ellas de manera arbitraria, por lo que inclusive es posible que se creen ciclos de manera que permite que la red tenga cierta memoria.
- Base Radial. Estas redes calculan la salida de la función con base a un punto denominado centro. La salida de estas redes es una combinación lineal de las funciones de activación utilizadas por las neuronas de manera individual.

También pueden clasificarse según el método de aprendizaje:

- Aprendizaje supervisado. El aprendizaje se realiza mediante un entrenamiento controlado por un "supervisor" que determina la respuesta correcta que debe generarse según cada entrada, de modo que si la salida no es correcta, modifica los pesos de las conexiones en la red con la finalidad de obtener una salida lo más cercana a la deseada.
- Aprendizaje no supervisado o autosupervisado. No requiere de un elemento externo que supervise los resultados, de modo que los pesos de cada conexión son determinados por la misma red.
- Aprendizaje por refuerzo. Utiliza probabilidades para ajustar los pesos de las conexiones, ya que no dispone de un conjunto completo de los datos exactos de salida.

2 Objetivo

Actualmente podemos encontrar diversas aplicaciones de las redes neuronales artificiales: clasificadores, detección de malware, anomalías de red, reconocimiento de voz, reconocimiento de imágenes, sistemas de recomendación, etc., pero, dado que las redes neuronales artificiales están inspiradas en el cerebro biológico, ¿pueden crear cosas como un humano?, por ejemplo, ¿son capaces de escribir un libro, crear una pintura o crear una pieza musical? Para este proyecto se

planea diseñar una red capaz de generar melodías musicales a partir de otras melodías de entrenamiento, buscando que estas melodías resulten agradables al ser escuchadas.

3 Metodología

Dado que la idea es diseñar una red neuronal que sea capaz de generar melodías musicales, debemos escoger con que herramientas trabajar. Sabemos que hay una amplia variedad de tipos de redes neuronales, así que hay que elegir una de acuerdo a las necesidades del problema, también debemos elegir con que tipo de formatos de música trabajar así como las herramientas que nos permitan manejar nuestro formato de música elegido.

A grandes rasgos podemos ver una pieza musical como un conjunto de datos secuenciales uno tras otro, por ello, para el tipo de red a utilizar podemos elegir una red de tipo recurrente que como sabemos, al contar con una memoria, es adecuada para trabajar con datos secuenciales, el problema de utilizar esta red es que su memoria es a corto plazo, por lo que para secuencias grandes este tipo de red no funciona de manera adecuada. Para solventar el problema de las redes recurrentes, se diseñaron las redes LSTM, que cuentan con una mayor memoria, por lo que este tipo de red es adecuada para trabajar con este problema.

Para el caso de los datos con los cuales trabajar, encontramos que existe una gran variedad de formatos para archivos musicales, entre ellos están el mp3, wma, wav, ogg, aac, raw, etc. Comparando los formatos encontramos que el más simple y sencillo entre ellos es el MIDI, por ello se decide trabajar con melodías en este formato.

Ya elegido el formato, se debe encontrar la manera de codificar los datos contenidos en un archivo MIDI. En la búsqueda de alguna herramienta que ayudara en esta tarea se encontró un conjunto de herramientas para python, desarrollado en el MIT (por el cuthbertLab, cuyo principal investigador es Michael Scott Cuthbert), llamado Music21, el cual se define como "un conjunto de herramientas basado en python para análisis musical asistido por computadora"[5]. Este conjunto de herramientas permite trabajar con archivos en formato MIDI, permitiendo extraer los datos que componen una canción en este formato de manera sencilla y posibilitando el crear también piezas musicales a partir de datos en forma de objetos que corresponden a las diversas componentes de una canción.

El conjunto de canciones base para el entrenamiento de la red se obtuvo de: https://www.vgmusic.com/music/other/miscellaneous/piano/, y consta de 9 canciones pertenecintes al soundtrack del videojuego "Chrono Trigger".

Ya establecida la red, el conjunto de datos con los que trabajar y las herramientas a utilizar, se pasa a la implementación del proyecto, proceso que consta de

cuatro apartados:

- **Codificación de los datos.**
  Se encontró que un archivo MIDI aún siendo simple cuenta con una gran cantidad de datos, entre los que considero más importantes están: notas, acordes, tonos, tiempos, silencios y tipo de instrumento. Al tratar de realizar la codificación de estos aspectos me encontré con que era complejo trabajar con todos estos datos, por lo que se elige mejor trabajar únicamente con piezas musicales de un solo instrumento (eligiendo el piano como instrumento), con notas y acordes.
  El código y la codificación resultante se muestran a continuación:

```
from music21 import converter, instrument, note, chord, pitch, stream
from keras.utils import np_utils
```

```
notas = []

codificar = []

for archivo in gl.glob("canciones/*.mid"):
    cancion = converter.parse(archivo)

    aux = instrument.partitionByInstrument(cancion)

    codificar = aux.parts[0].recurse()

    for elemento in codificar:
        if isinstance(elemento, note.Note):
            n = str(elemento.pitch)
            notas.append(n)
        if isinstance(elemento, chord.Chord):
            notes = []
            for nota in elemento:
                notes.append(str(nota.pitch.midi))
            a = '.'.join(notes)
            notas.append(a)
```

- Notas:
  - A2, B6, D3,...
- Acordes:
  - A2.D3.B2
- Ejemplo:
  - [A2, B6, C3.D4.E2, E2...]

- **Creación de secuencias para el entrenamiento.**
  Lo que se hace básicamente es crear subsecuencias de la secuencia inicial generada anteriormente y de tamaño definido (en el proyecto utilizo secuencias de tamaño 100), donde la nota a predecir es la siguiente nota después de la última en la subsecuencia, el código y el proceso de generación de subsecuencias son:

```
num_notas_distintas = len(set(notas))
tam_secuencia = 100

tonos = sorted(set(elemento for elemento in notas))
nota_a_int = dict((nota, numero) for numero, nota in enumerate(tonos))
int_a_nota = dict((numero, nota) for numero, nota in enumerate(tonos))
```

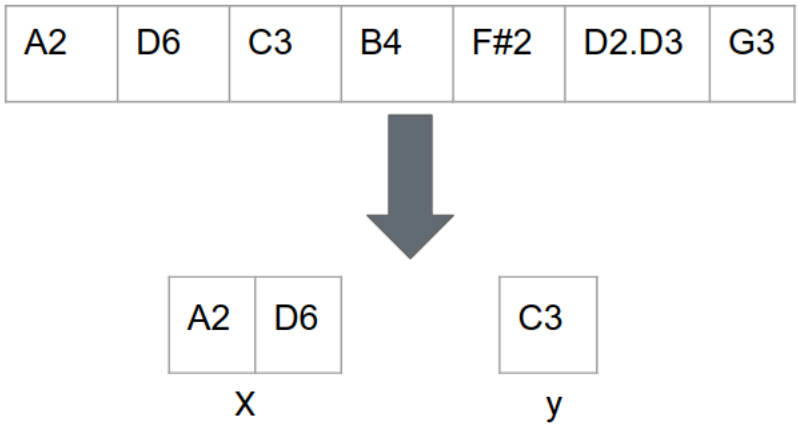
```
X = []
y = []

for i in range(0, len(notas) - tam_secuencia, 1):
    secuencia = notas[i: i+tam_secuencia]
    nota_sig = notas[i + tam_secuencia]
    X.append([nota_a_int[nota] for nota in secuencia])
    y.append(nota_a_int[nota_sig])
```

```
X_tr = np.reshape(X, (len(X), tam_secuencia, 1))


X_tr = X_tr / num_notas_distintas


y_tr = np_utils.to_categorical(y)
```

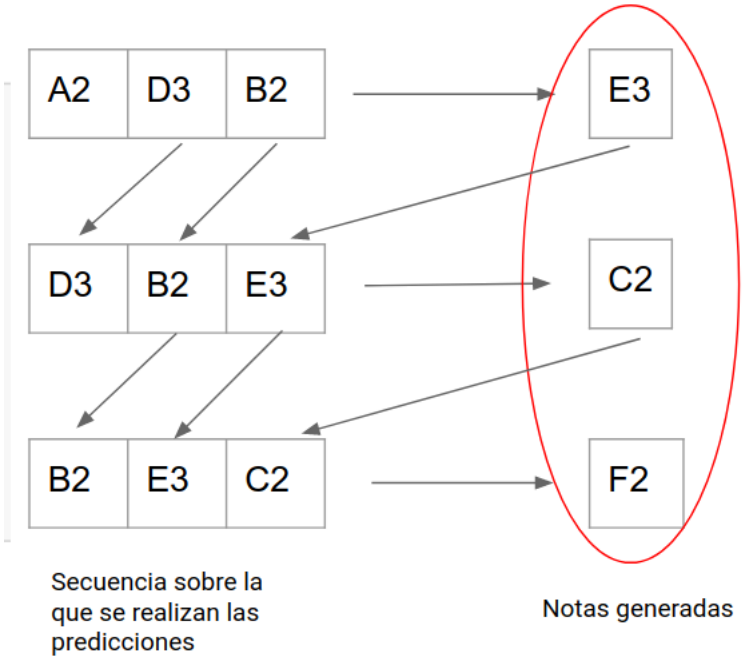


- **Creación y entrenamiento de la red.**
  La arquitectura final de la red se muestra a continuación:

Layer (type)	Output Shape	Param #
lstm_4 (LSTM)	(None, 100, 512)	1052672
lstm_5 (LSTM)	(None, 100, 512)	2099200
lstm_6 (LSTM)	(None, 512)	2099200
batch_normalization_3 (Batch	(None, 512)	2048
dropout_3 (Dropout)	(None, 512)	0
dense_3 (Dense)	(None, 256)	131328
activation_3 (Activation)	(None, 256)	0
batch_normalization_4 (Batch	(None, 256)	1024
dropout_4 (Dropout)	(None, 256)	0
dense_4 (Dense)	(None, 451)	115907
activation_4 (Activation)	(None, 451)	0

Total params: 5,501,379
Trainable params: 5,499,843
Non-trainable params: 1,536

- **Creación de melodía a partir de la red entrenada.**
  Lo primero que se hace es elegir de manera aleatoria una secuencia inicial sobre la que la red irá generando las notas/acordes que conformarán la melodía. A partir de la secuencia inicial, la red predice la siguiente nota a esta secuencia, la nota resultante será la primera nota de la canción generada (la secuencia sobre la que se realizan las predicciones no es parte de la medlodía resultante)y también esta nota se agregará a la secuencia sobre la que se van realizando las predicciones, el proceso se puede ver en la siguiente figura:



El código utilizado para esta parte es el siguiente:

```
inicio = np.random.randint(0, len(X_tr-1))
secuencia = X[inicio]
composicion = []

# generamos notas
for indice in range(500):
    prediccion_in = np.reshape(secuencia, (1, len(secuencia), 1))
    prediccion_in = prediccion_in / float(num_notas_distintas)

    prediccion = model.predict(prediccion_in, verbose=0)

    index = np.argmax(prediccion)
    result = int_a_nota[index]
    composicion.append(result)

    secuencia.append(index)
    secuencia = secuencia[1:len(secuencia)]
```

Una vez generadas las notas/acordes que contendrá la melodía, es momento de generar el archivo MIDI para poder escuchar el resultado, para esto, dado que la red solo contiene notas y acordes, se establece que cada una de estas componentes tendrá una duración (offset) de 0.5, es decir, todas las notas y acordes tendrán la misma duración, una vez hecho esto, se realiza el proceso inverso de la codificación para poder obtener objetos de tipo Note Y Chord de music21, de este modo es posible generar el archivo MIDI.
El código utilizado es el siguiente:

```
offset = 0
cancion = []

for elemento in composicion:

    if ('.' in elemento) or elemento.isdigit():
        notas_acorde = elemento.split('.')
        notes = []
        for nota_actual in notas_acorde:
            new_nota = note.Note(int(nota_actual))
            new_nota.storedInstrument = instrument.Piano()
            notes.append(new_nota)
        new_acorde = chord.Chord(notes)
        new_acorde.offset = offset
        cancion.append(new_acorde)
    else:
        new_nota = note.Note(elemento)
        new_nota.offset = offset
        new_note.storedInstrument = instrument.Piano()
        cancion.append(new_note)

    offset += 0.5

midi_stream = stream.Stream(cancion)

midi_stream.write('midi', fp='cancion.mid')
```

4 Resultados

Se adjuntan algunas melodías resultantes de la red, también se adjunta el modelo utilizado para generar estas melodías (archivo "musica.h5").
A grandes rasgos se aprecia que las melodías resultantes si son agradables al ser escuchadas, aunque debido a la restricción sobre el tiempo de cada nota/acorde, las melodías son muy "cuadradas" en el sentido que no se escuchan variaciones de duración de los sonidos.
Se incluye una melodía ("cancion11.mid") resultante del entrenamiento con otro dataset ("canciones1.h5"), el cual fue obtenido de: https://www.virtualsheetmusic.com/downloads/HalloweenMidi.html. La intención de este dataset era pro-

ducir canciones de "terror", lo cual a grandes rasgos se cumple, aunque las melodías resultantes resultaron monótonas, ya que se la pasan realizando muchas repeticiones de las mismas secuencias.

5 Conclusión

Para finalizar el trabajo creo que se cumplió el objetivo propuesto del proyecto, ya que la red neuronal desarrollada si es capaz de generar nuevas melodías musicales a partir de las melodías base.
En cuanto a la red, después de realizar algunas variaciones encontré que la arquitectura propuesta funcionó de manera adecuada, aunque el tiempo computacional es elevado. Para el entrenamiento final que consta de 50 épocas, la red tardó cerca de 8 hrs, por lo que al agregar más capas a la red o al utilizar un mayor conjunto de canciones base este tiempo aumenta de manera considerable.
También encontré que cuanta más variedad musical tengan las canciones base del entrenamiento, los resultados se escuchan mejor (desde mi apreciación), mientras que si las canciones base son parecidas entre sí, el resultado son melodías monótonas, donde se repiten muchas veces las mismas partes. (Esto se observa al entrenar la red con canciones de terror, donde resultan melodías algo monótonas, mientras que con las canciones de Chorno Trigger que tienen mayor variedad, los resultados también tienen mayor variedad).
Como una gran mejora para los resultados, creo que sería necesario agregar los tiempos de duración de las notas/acordes, de manera que la red sea capaz de generar melodías donde los tiempos de duración de cada elemento serían variados, y de esta manera las melodías serían más ricas musicalmente hablando.

References

[1] Khepri,William. Redes Neuronales, ¿qué son?. 2018.
https://medium.com/@williamkhepri/redes-neuronales-que-son-a64d022298e0

[2] Calvo, Diego. Clasificación de redes neuronales artificiales. 2017.

[3] Olah, Christopher. Understanding LSTM Networks. 2015.
https://colah.github.io/posts/2015-08-Understanding-LSTMs/
https://www.diegocalvo.es/clasificacion-de-redes-neuronales-artificiales/

[4] https://web.mit.edu/music21/doc/about/what.html

[5] Documentación de Music21
https://web.mit.edu/music21/doc/index.html