Gerar Música Usando LSTM com Keras

Docente: Jean Pierre Briot
Discente: Vinícius Condina

Introdução

O artigo apresenta um tutorial de como criar música utilizando redes neurais concorrentes com a Biblioteca Keras

Background

Redes Neurais Recorrentes(RNN):

Classe de redes neurais que utilizam informação sequencial. Os cálculos são dependentes dos anteriores.

Long Short-Term Memory(LSTM):

Tipo de RNN, são úteis para reconhecer padrões de longo prazo, utilizando mecanismos de gate.

Background

Music21:

Kit de ferramentas Python para musicologia. Permite criar notas e acordes para a criação de músicas. Além de adquirir notação de arquivos MIDI.

Keras:

API de rede neural que simplifica interação com TensorFlow. No artigo é utilizado para criar e treinar o LSTM. Após isto, para criar a notação musical.

Dados

Músicas de Piano extraídas grande parte do Final Fantasy. O primeiro passo da geração de música é examinar os dados que estão sendo trabalhados.

```
<music21.note.Note F>
<music21.chord.Chord A2 E3>
<music21.chord.Chord A2 E3>
<music21.note.Note E>
<music21.chord.Chord B-2 F3>
<music21.note.Note F>
<music21.note.Note G>
<music21.note.Note D>
<music21.chord.Chord B-2 F3>
<music21.note.Note F>
<music21.chord.Chord B-2 F3>
<music21.note.Note E>
<music21.chord.Chord B-2 F3>
<music21.note.Note D>
<music21.chord.Chord B-2 F3>
<music21.note.Note E>
<music21.chord.Chord A2 E3>
```

Tipos de Objetos

Note:

Pitch - Frequência do som, sendo A o mais alto e G o menos

Octave - Conjunto de Pitches do Piano

Chords:

Conjunto de notas tocadas ao mesmo tempo

Dados

Para gerar com precisão, a rede neural deve ser capaz de prever a próxima nota ou acorde.

Neste treinamento, foram utilizados 352 acordes e notas. Agora, é necessário verificar o intervalo das notas através do M21.

```
<music21.note.Note B> 72.0
<music21.chord.Chord E3 A3> 72.0
<music21.note.Note A> 72.5
<music21.chord.Chord E3 A3> 72.5
<music21.note.Note E> 73.0
<music21.chord.Chord E3 A3> 73.0
<music21.chord.Chord E3 A3> 73.5
<music21.note.Note E-> 74.0
<music21.chord.Chord F3 A3> 74.0
<music21.chord.Chord F3 A3> 74.5
<music21.chord.Chord F3 A3> 75.0
<music21.chord.Chord F3 A3> 75.5
<music21.chord.Chord E3 A3> 76.0
<music21.chord.Chord E3 A3> 76.5
<music21.chord.Chord E3 A3> 77.0
<music21.chord.Chord E3 A3> 77.5
<music21.chord.Chord F3 A3> 78.0
<music21.chord.Chord F3 A3> 78.5
<music21.chord.Chord F3 A3> 79.0
```

Preparando os Dados

Primeiro passo é armazenar em um Array:

```
from music21 import converter, instrument, note, chord

notes = []

for file in glob.glob("midi_songs/*.mid"):
    midi = converter.parse(file)
    notes_to_parse = None

    parts = instrument.partitionByInstrument(midi)

    if parts: # file has instrument parts
        notes_to_parse = parts.parts[0].recurse()
    else: # file has notes in a flat structure
        notes_to_parse = midi.flat.notes

    for element in notes_to_parse:
        if isinstance(element, note.Note):
            notes.append(str(element.pitch))
        elif isinstance(element, chord.Chord):
            notes.append('.'.join(str(n) for n in
element.normalOrder))
```

Preparando os Dados

Em seguida, cria sequências de entradas e suas saídas

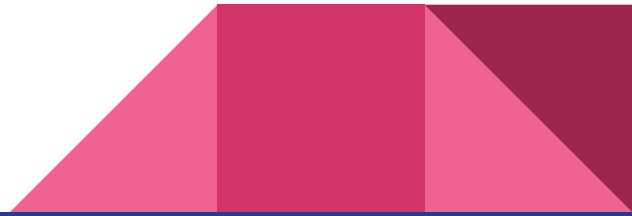
```
sequence_length = 100

# get all pitch names
pitchnames = sorted(set(item for item in notes))

# create a dictionary to map pitches to integers
note_to_int = dict((note, number) for number, note in
enumerate(pitchnames))

network_input = []
network_output = []

# create input sequences and the corresponding outputs
for i in range(0, len(notes) - sequence_length, 1):
    sequence_in = notes[i:i + sequence_length]
    sequence_out = notes[i + sequence_length]
    network_input.append([note_to_int[char] for char in
sequence_in])
    network_output.append(note_to_int[sequence_out])

n_patterns = len(network_input)

# reshape the input into a format compatible with LSTM layers
network_input = numpy.reshape(network_input, (n_patterns,
sequence_length, 1))
# normalize input
network_input = network_input / float(n_vocab)

network_output = np_utils.to_categorical(network_output)
```

Modelo

Quatro camadas são usadas no modelo:

LSTM Layer: Camada que toma uma sequência como entrada, podendo retornar as sequências ou uma Matriz.

Dropout Layer: Camada de descarte que define entrada de 0 a cada atualização para prevenir Overfitting.



Modelo

Dense Layer: Camada de conexão entre nós de entrada e de saída.

Activation Layer: Determina qual função de ativação a RN irá utlizar

Para LSTM, Dense e Activation, o primeiro parâmetro é a quantidade de nós que a camada vai ter.

MODELO

```
model = Sequential()
    model.add(LSTM(
        256,
        input_shape=
(network_input.shape[1],
network_input.shape[2]),
        return_sequences=True
    ))
    model.add(Dropout(0.3))
    model.add(LSTM(512,
return_sequences=True))
    model.add(Dropout(0.3))
    model.add(LSTM(256))
    model.add(Dense(256))
    model.add(Dropout(0.3))
    model.add(Dense(n_vocab))

model.add(Activation('softmax'))

model.compile(loss='categorical_c
rossentropy',
optimizer='rmsprop')
```

Para a primeira camada, temos que fornecer um parâmetro exclusivo chamado input_shape. O objetivo do parâmetro é informar à rede a forma dos dados que serão treinados.

A última camada deve sempre conter a mesma quantidade de nós que o número de saídas diferentes que o nosso sistema possui. Isso garante que a saída da rede seja mapeada diretamente para nossas classes.

Treinamento

Função model.fit é utilizada para treinar a rede. Nesta, treinando por 200 iterações e com tamanho de amostragem de 64.

```
filepath = "weights-improvement-{epoch:02d}-{loss:.4f}-bigger.hdf5"

checkpoint = ModelCheckpoint(
    filepath, monitor='loss',
    verbose=0,
    save_best_only=True,
    mode='min'
)
callbacks_list = [checkpoint]

model.fit(network_input, network_output, epochs=200, batch_size=64,
callbacks=callbacks_list)
```

Gerando Música

Após treinar, vamos para a Geração!

Como temos uma lista completa de sequências de anotações à nossa disposição, escolheremos um índice aleatório na lista como ponto de partida, o que nos permite executar novamente o código de geração sem alterar nada e obter resultados diferentes todas as vezes

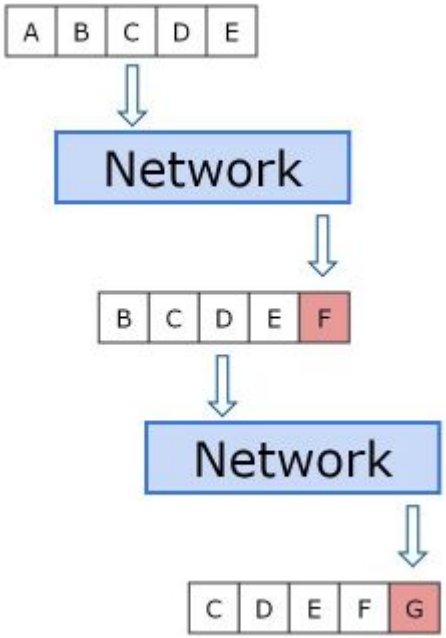
Gerando Música

```
start = numpy.random.randint(0, len(network_input)-1)

int_to_note = dict((number, note) for number, note in
enumerate(pitchnames))

pattern = network_input[start]
prediction_output = []

# generate 500 notes
for note_index in range(500):
    prediction_input = numpy.reshape(pattern, (1, len(pattern), 1))
    prediction_input = prediction_input / float(n_vocab)

    prediction = model.predict(prediction_input, verbose=0)

    index = numpy.argmax(prediction)
    result = int_to_note[index]
    prediction_output.append(result)

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

Escolhemos gerar 500 notas usando a rede, já que são aproximadamente dois minutos de música e dá à rede bastante espaço para criar uma melodia.

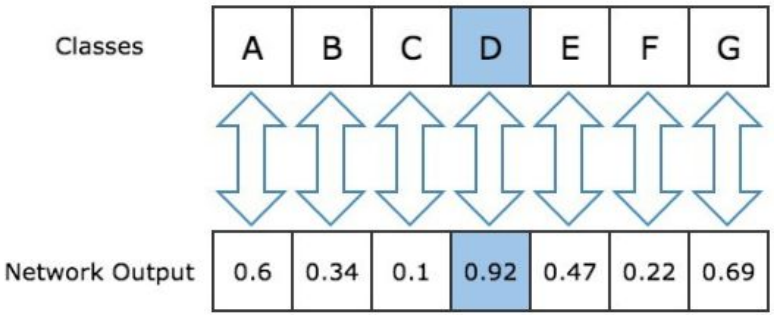
Para cada nota que queremos gerar, temos que enviar uma sequência para a rede.

Gerando Música



A primeira sequência que apresentamos é a sequência de notas no índice inicial. Para cada sequência subseqüente que usamos como entrada, removeremos a primeira nota da sequência e inseriremos a saída da iteração anterior no final da sequência, como pode ser visto na figura.

Gerando Música



Para determinar a previsão mais provável da saída da rede, extraímos o índice do valor mais alto. O valor no índice X na matriz de saída corresponde à probabilidade de que X seja a próxima nota.

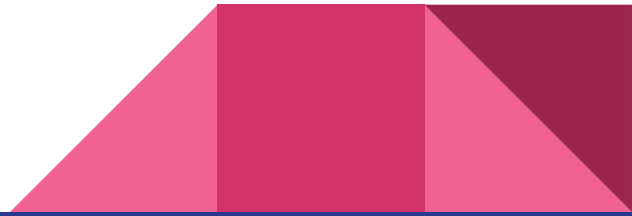
Gerando Música

Em seguida, coletamos todas as saídas da rede em um único array.

Agora, temos que determinar se as saídas são chords ou notes.

Se o padrão é um Chord, temos que dividir a string em uma matriz de notas. Em seguida, percorremos a representação de string de cada nota e criamos um objeto Note para cada uma delas.

Se o padrão for uma nota, criamos um objeto Note usando a representação de string do pitch.



Gerando Músicas

```
offset = 0
output_notes = []

# create note and chord objects based on the values generated by the
model

for pattern in prediction_output:
    # pattern is a chord
    if ('.' in pattern) or pattern.isdigit():
        notes_in_chord = pattern.split('.')
        notes = []
        for current_note in notes_in_chord:
            new_note = note.Note(int(current_note))
            new_note.storedInstrument = instrument.Piano()
            notes.append(new_note)
        new_chord = chord.Chord(notes)
        new_chord.offset = offset
        output_notes.append(new_chord)
    # pattern is a note
    else:
        new_note = note.Note(pattern)
        new_note.offset = offset
        new_note.storedInstrument = instrument.Piano()
        output_notes.append(new_note)

    # increase offset each iteration so that notes do not stack
    offset += 0.5
```

Gerando Música

Agora, com a lista de chords e notes criada, iremos criar o arquivo MIDI pelo M21

```
midi_stream = stream.Stream(output_notes)

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

Resultados

Música gerada

Trabalhos Futuros e Melhorias

Primeiro, a implementação que temos no momento não suporta duração variável de notas e diferentes deslocamentos entre notas.

Outra falha é que a rede não tem começo e fim, a música é encerrada abruptamente.

A rede também entraria em estado de falha se entrasse uma nota que ela não conhecesse.

Bibliografia

https://towardsdatascience.com/how-to-generate-music-using-a-lstm-neural-network-in-keras-68786834d4c5

https://soundcloud.com/sigur-ur-sk-li/neuralnet-music-1?in=sigur-ur-sk-li/sets/music-generated-by-a-neural-network

https://en.wikipedia.org/wiki/Long_short-term_memory

http://colah.github.io/posts/2015-08-Understanding-LSTMs/