UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
INSTITUTO DE COMPUTAÇÃO
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

ALINE FREIRE DE REZENDE

EXPERIMENTO EM COMPOSIÇÃO DE SONATAS BEETHOVENIANAS ATRAVÉS DE REDES NEURAIS LSTM

RIO DE JANEIRO
2023

ALINE FREIRE DE REZENDE

EXPERIMENTO EM COMPOSIÇÃO DE SONATAS BEETHOVENIANAS ATRAVÉS DE REDES NEURAIS LSTM

Trabalho de conclusão de curso de graduação apresentado ao Instituto de Computação da Universidade Federal do Rio de Janeiro como parte dos requisitos para obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Prof. João Carlos Pereira da Silva
Co-orientador: Prof. Liduino José Pitombeira de Oliveira

RIO DE JANEIRO
2023

CIP - Catalogação na Publicação

R467e

Rezende, Aline Freire de
Experimento em composição de sonatas beethovenianas através de redes neurais LSTM / Aline Freire de Rezende. -- Rio de Janeiro, 2023.
58 f.

Orientador: João Carlos Pereira da Silva.
Coorientador: Liduino José Pitombeira de Oliveira.
Trabalho de conclusão de curso (graduação) - Universidade Federal do Rio de Janeiro, Instituto de Computação, Bacharel em Ciência da Computação, 2023.

1. Rede neural LSTM. 2. Composição por IA. 3. Sonatas de Beethoven. I. Silva, João Carlos Pereira da, orient. II. Oliveira, Liduino José Pitombeira de, coorient. III. Título.

Elaborado pelo Sistema de Geração Automática da UFRJ com os dados fornecidos pelo(a) autor(a), sob a responsabilidade de Miguel Romeu Amorim Neto - CRB-7/6283.

ALINE FREIRE DE REZENDE

EXPERIMENTO EM COMPOSIÇÃO DE SONATAS BEETHOVENIANAS ATRAVÉS DE REDES NEURAIS LSTM

Trabalho de conclusão de curso de graduação apresentado ao Instituto de Computação da Universidade Federal do Rio de Janeiro como parte dos requisitos para obtenção do grau de Bacharel em Ciência da Computação.

Aprovado em 31 de Agosto de 2023

BANCA EXAMINADORA:

Documento assinado digitalmente
gov.br
JOAO CARLOS PEREIRA DA SILVA
Data: 06/09/2023 08:02:19-0300
Verifique em https://validar.iti.gov.br

Prof. João Carlos Pereira da Silva, D.Sc. (UFRJ)

Documento assinado digitalmente
gov.br
LIDUINO JOSE PITOMBEIRA DE OLIVEIRA
Data: 06/09/2023 09:56:20-0300
Verifique em https://validar.iti.gov.br

Prof. Liduino José Pitombeira de Oliveira, PhD. (UFRJ)

Documento assinado digitalmente
gov.br
CAROLINA GIL MARCELINO
Data: 06/09/2023 14:30:45-0300
Verifique em https://validar.iti.gov.br

Profa. Carolina Gil Marcelino, D.Sc. (UFRJ)

Documento assinado digitalmente
gov.br
JOAO ANTONIO RECIO DA PAIXAO
Data: 07/09/2023 09:34:49-0300
Verifique em https://validar.iti.gov.br

Prof. João Antonio Recio Paixão, D.Sc. (UFRJ)

Documento assinado digitalmente
gov.br
PEDRO FARIA PROENCA GOMES
Data: 06/09/2023 10:23:09-0300
Verifique em https://validar.iti.gov.br

Pedro Faria Proença Gomes, M.M. (UFRJ)

Dedico esta pesquisa às pessoas que me apoiaram emocionalmente durante o desenvolvimento, em especial meus pais, Heloisa Machado Freire e Marcelo Marota de Rezende; meus amigos Gilberto Lopes, Letícia Tavares, Marcelo Godoy, Mayara Martins, Tainah Martins e Yguaciara Mello; e minha prima Giulia Freire. Dedico também aos meus cachorros Caco, Luppi, Lycan e Mel por serem como raios de sol nos momentos difíceis ao decorrer do processo.

AGRADECIMENTOS

Agradeço ao professor João Carlos Pereira da Silva do Instituto de Computação da UFRJ, e ao professor Liduino José Pitombeira de Oliveira da Escola de Música da UFRJ pela orientação durante o desenvolvimento deste trabalho. Também agradeço ao aluno de doutorado do professor Liduino, Pedro Proença, por contribuir com materiais práticos para o desenvolvimento do projeto. E por fim, agradeço ao aluno Gilberto Lopes, à aluna Letícia Tavares e à ex-aluna Mayara Martins, todos do Instituto de Computação da UFRJ, pelo apoio na elaboração de programas e textos ao longo do processo.

"The true artist is not proud, he unfortunately sees that art has no limits; he feels darkly how far he is from the goal; and though he may be admired by others, he is sad not to have reached that point to which his better genius only appears as a distant, guiding sun."

Ludwig van Beethoven

RESUMO

Devido a avanços tecnológicos no último século, tornou-se possível a experimentação de composições musicais por meio de inteligência artificial e redes neurais. Dentre as inúmeras possibilidades de formas musicais, este trabalho buscou replicar composições de sonatas para piano, em especial beethovenianas, por meio de redes neurais do tipo LSTM (Long Short Term Memory-Memória de Curto Prazo Longa). Inicialmente foram estudados modos de coleta e diferentes formas de representação das informações musicais. Em sequência, foram exploradas redes LSTM com arquiteturas variantes, tanto na quantidade de camadas quanto nos parâmetros de entrada da rede. E por fim, foram executadas algumas tentativas de construção de peças musicais para fins de análise. Dentre os resultados gerados, não foi possível gerar peças do tipo sonata que apresentassem qualidade musical condizente, o que pode ter ocorrido pela quantidade de informações utilizadas e a insuficiência de recursos computacionais necessários para que o aprendizado ocorresse de forma desejada.

Palavras-chave: inteligência artificial; redes neurais; lstm; música; beethoven; composição por ia; python.

ABSTRACT

Due to the technological advances in the last century, attempting to compose music through artificial intelligence and neural networks has become possible. Among the countless possibilities of musical forms, this work focused on replicating piano sonata compositions, especially from Beethoven, through LSTM (Long Short Term Memory) neural networks. First, different ways of collecting and representing the desired musical information were studied. Then, LSTM networks with variant architectures were explored, both in the number of layers and in the network input parameters. Finally, a reconstruction of the musical piece was executed for analysis purposes. Among the generated results, it was not possible to generate sonata-type pieces that showed consistent musical quality, and this may have occurred due to the large amount of information used and the lack of available computational resources necessary for the network's learning to happen in the desired way.

Keywords: artificial intelligence; neural networks; lstm; music; beethoven; ai composition; python.

LISTA DE ILUSTRAÇÕES

LISTA DE CÓDIGOS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Dentre os inúmeros compositores clássicos que tivemos, um dos mais notórios foi Ludwig van Beethoven (Figura 1) que deixou à humanidade cerca de 200 obras, como sonatas, sinfonias, concertos, quartetos para cordas, bagatelas, entre outras. Após sua morte, foi encontrado em seus itens pessoais, um caderno de composições (GERTSCH, 2015). Nele havia algumas obras nunca publicadas, e algumas incompletas. Com muito potencial, surge-se a indagação de como completar suas peças, ou até mesmo criar novas peças, seguindo características beethovenianas. Uma forma de se resolver esta problemática pode talvez ser utilizando o aprendizado de máquina, para que seja possível captar padrões e peculiaridades do compositor.

Figura 1 – Retrato de Ludwig van Beethoven por Joseph Karl Stieler, 1820. Fonte: https://commons.wikimedia.org/wiki/File:Beethoven_1820.jpg. Acesso em 24/10/2022.

O aprendizado de máquina é um campo da ciência da computação que derivou-se do estudo da modelagem matemática de redes neurais, onde tentava-se modelar a lógica do pensamento e tomada de decisões humanos, por volta de 1943, quando Walter Pitts e Warren McCulloch (Figura 2) publicaram um artigo sobre o tema (NIELSEN, 2018). Programas foram desenvolvidos que implicavam que o computador seria capaz de tomar decisões, em especial, em jogos como o xadrez. Os cientistas McCulloch e Pitts propuseram o primeiro neurônio artificial (MCCULLOCH; PITTS, 1943), o chamado "Perceptron", onde é recebida uma informação, e esse neurônio é capaz de usar pesos para calcular uma saída binária. A partir deste marco, foi possível inovar o aprendizado de máquina com

o passar dos anos, até chegarmos aos dias atuais, onde o utilizamos em diversas áreas, tais como processamento de linguagem natural, manipulação textual, reconhecimento de imagens, criação musical, entre outros.

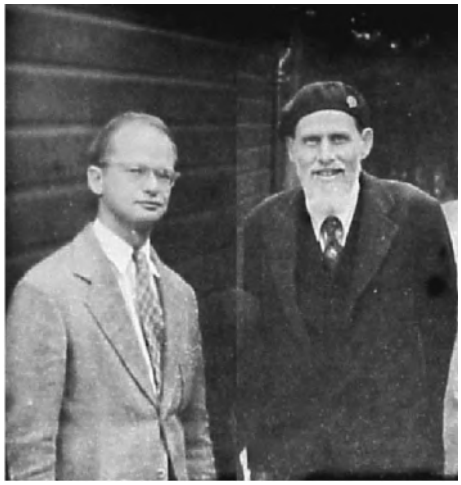

Figura 2 – Retrato de Warren McCulloch (direita) e Walter Pitts (esquerda) em 1949. Fonte: https://www.historyofinformation.com/detail.php?entryid=782. Acesso em 11/07/2023.

A união entre computação e composição musical já existe desde 1957, quando um software utilizado por Newman Guttman gerou uma melodia de 17 segundos intitulada "*The Silver Scale*"[1] (BRIOT; HADJERES; PACHET, 2020). Desde então, houve pesquisas para inovar os estudos de composição algorítmica. Na década de 60, Iannis Xenakis (Figura 3) se voltou a estudar composição estocástica, por meio de modelos markovianos[2] (XENAKIS, 1963). Na década de 80, David Cope (Figura 4), um importantíssimo cientista e compositor, se dedicou à área de inteligência artificial musical, criando um sistema chamado *Experiments in Musical Intelligence* (EMI), capaz de, através de um conjunto de outras músicas, aprender suas regras, padrões, e criar sua própria gramática[3] (COPE, 1996).

Além dessas pesquisas, (MUÑOZ-LAGO; MÉNDEZ, 2020) experimentaram gerar a 10ª Sinfonia de Beethoven, executando o treinamento com as 9 Sinfonias compostas por Beethoven, e utilizando redes neurais do tipo LSTM (*Long Short-Term Memory*), e (CRESTEL; ESLING, 2016) experimentaram a geração de música orquestral em tempo real, utilizando as seguintes redes neurais condicionais: RBM (*Restricted Boltzmann machine*), cRBM (*Convolutional Restricted Boltzmann machine*) and FGcRBM (*Factored-Gated Convolutional Restricted Boltzmann machine*)[4].

[1] Áudio disponível em https://www.youtube.com/watch?v=PM64-lqYyZ8
[2] Um exemplo de peça gerada está disponível em https://www.youtube.com/watch?v=5qS5lqbx9H0
[3] Um exemplo de peça gerada está disponível em https://www.youtube.com/watch?v=2kuY3BrmTfQ
[4] Seu banco de dados de arquivos MIDI pode ser encontrado em https://qsdfo.github.io/LOP/database.

Figura 3 – Retrato de Iannis Xenakis em 1977. Fonte: http://120years.net/upic-system-iannis-xenakis-france-1977/. Acesso em 19/07/2023.

Figura 4 – Retrato de David Cope trabalhando no EMI. Fonte: https://computerhistory.org/blog/algorithmic-music-david-cope-and-emi/. Acesso em 11/07/2023.

Atualmente, com a ajuda dos avanços tecnológicos, as ferramentas necessárias para investir na união dessas áreas de conhecimento se tornou mais acessível, contando com contribuições do público comum, não-acadêmico, para o avanço da composição algorítmica. É o caso de Sigurður Skúli, que experimentou replicar músicas de *video-games*[5] também utilizando redes LSTM (SKÚLI, 2017).

Tendo em vista a indagação inicial e as possibilidades que o aprendizado de máquina nos disponibiliza, o objetivo deste trabalho é estudar a possibilidade de gerar novas peças por meio de aprendizado de máquina, especialmente utilizando redes neurais, e analisar se o resultado desse processo é capaz de reproduzir as idiossincrasias do estilo composição de Beethoven. Além disso, dentre o vasto portifólio de Beethoven, escolhemos as suas sonatas para piano. Escolha essa motivada pela presença e importância da Sonata nº 8 - Pathétique em C maior[6] na vida pessoal da autora deste trabalho.

Inicialmente, mapeamos os possíveis problemas que o tipo de dado escolhido poderia

[5] Peças geradas disponíveis em https://soundcloud.com/sigur-ur-sk-li/neuralnet-music-1?in=sigur-ur-sk-li/sets/music-generated-by-a-neural-network.

[6] Disponível em https://www.youtube.com/watch?v=f0LgGt-XR8o

trazer para o desenvolvimento, e com isso estabelecemos, dentro das ferramentas disponíveis do aprendizado de máquina, o uso da rede neural LSTM (*Long Short-Term Memory*) buscando uma forma de minimizar os problemas em questão. No Capítulo 2 definimos conceitos teóricos.

Em seguida, decidimos os atributos musicais essenciais para designar as possíveis características de Beethoven, e descrevemos cada um deles no Capítulo 3.

Durante o desenvolvimento do projeto, situado no Capítulo 4, pusemos em prática os conhecimentos adquiridos com suas particularidades, e preparamos os dados e executamos o treinamento da rede neural com as variações necessárias para que ela pudesse concluir sua tarefa e gerar uma saída analisável, tanto do ponto de vista computacional quanto musical.

2 APRENDIZADO DE MÁQUINA

O aprendizado de máquina é um campo da inteligência artificial que utiliza algoritmos que permitem construir modelos cujo objetivo é realizar uma determinada tarefa. Há essencialmente 4 tipos de abordagem (IBM, 2022):

a) Aprendizado supervisionado: Neste caso, o programa aprendiz recebe um conjunto formado pelo par entrada e saída desejada. Busca-se um modelo baseado no conjunto de dados conhecidos.

b) Aprendizado não-supervisionado: Já neste caso, não há um conjunto de saídas desejadas. O programa é responsável por categorizá-las por meio de seu aprendizado, encontrando padrões e características.

c) Aprendizado semi supervisionado: Um tipo de mistura dos dois casos anteriores, onde algumas entradas têm suas saídas desejadas faltando, e o programa precisa tanto buscar uma regra geral quanto categorizar as faltantes.

d) Aprendizado por reforço: Por fim, neste caso o programa atua em um ambiente dinâmico. Conforme ele opera, é fornecido a ele alguma forma de retorno positivo ou negativo para que ele mesmo consiga se ajustar ainda durante a execução da dada tarefa.

No aprendizado de máquina supervisionado, não-supervisionado e semi supervisionado utiliza-se um grande conjunto de dados para treino cujo resultado final é conhecido. A partir disto, o programa em questão tem a capacidade de melhorar sua performance de executar uma certa tarefa comparando o seu resultado com o resultado esperado. Ao final de uma iteração, chamada de época, ele ajusta os pesos de seu processo de predição e tenta executar a mesma tarefa novamente.

2.1 APRENDIZADO PROFUNDO

O aprendizado profundo é um dos métodos do aprendizado de máquina, baseado em redes neurais, conceito inspirado na biologia neurológica humana (NIELSEN, 2018). Este tipo de técnica pode empregar abordagens de aprendizado supervisionado, não-supervisionado ou semi supervisionado. O aprendizado profundo já se mostrou ser uma solução viável para problemas relacionados a reconhecimento de imagem, processamento de fala e áudio, processamento de linguagens naturais, robótica, bioinformática e química, jogos virtuais, mecanismos de busca, anúncios online e finanças (GOODFELLOW; BENGIO; COURVILLE, 2016).

Como redes neurais se inspiram na nossa biologia, faz-se necessário entender que o cérebro humano tem capacidades mais avançadas em relação à tomada de decisão, se

comparadas às de um programa de computador dos tempos atuais. Um clássico exemplo de problema a ser resolvido que evidencia esta diferença é a tarefa de se reconhecer um número escrito à mão pela sua imagem (NIELSEN, 2018).

Definir regras para esse tipo de problema se mostra complicado, pois é possível se perder em exceções. Então a forma que as redes neurais funcionam é considerando um conjunto para treinamento e assim desenvolve-se um sistema que consegue aprender através de um conjunto de exemplos de treinamento. Quanto maior o conjunto de treino, mais acurada é a capacidade de aprender do computador, já que há mais exemplos de onde ele poderá extrair padrões.

2.1.1 Arquitetura de Redes Neurais

Uma rede neural artificial (Figura 5) é composta por uma camada de entrada, contendo nós iniciais; uma camada de saída, contendo os nós finais; e uma camada intermediária, chamada camada escondida (*hidden layer*), apenas indicando que não são nem entrada e nem saída. É possível ter inúmeras camadas escondidas, e quando é o caso, são chamadas de Percéptrons Multi-camadas (ou MLP, do inglês *Multilayer Perceptrons*). Camadas escondidas são mais complexas de se definir. Quando a saída de uma camada vira entrada da camada seguinte, não havendo *loops*, nós temos o que é chamado de rede neural *feed-forward* (NIELSEN, 2018). Redes neurais estruturadas em muitas camadas são chamadas de **redes neurais profundas**.

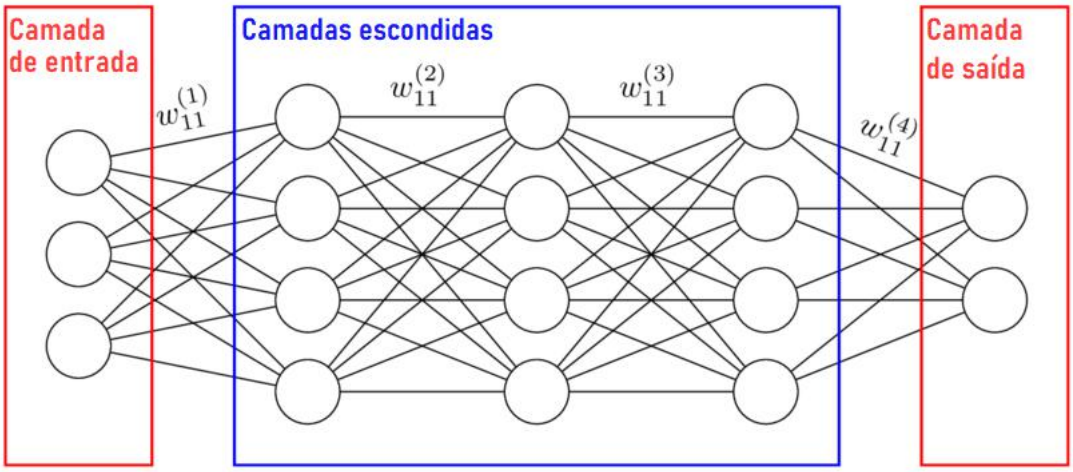

Figura 5 – Camadas de entrada, saída e intermediárias de uma rede neural, onde os ω's representam o peso de cada aresta entre os nós - Fonte: https://images.deepai.org/glossary-terms/4c9d8f89916848b4803df475ef6892be/hiddenlayer.png. Acesso em 14/01/2023.

Uma solução que as próprias redes neurais tomam no meio do processo, internamente, é dividir o problema em subproblemas. Por exemplo, podemos identificar que uma imagem

contém um rosto, caso seja possível identificar os elementos que o compõe, como olhos, nariz e boca.

O processo básico que uma rede neural executa é utilizar parte do conjunto de dados de exemplo para treinar a execução da tarefa, e outra parte para prever o resultado final, tendo um modelo construído e testado para que se preveja um resultado final. Durante o treinamento, utilizamos como entrada para essa rede um grande volume de exemplos, para que ela possa tentar prever o próximo passo e ter uma espécie de gabarito. Nessa fase, utilizamos uma **função de perda** para quantificar a melhora do aprendizado da rede, buscando sempre reduzir a perda a cada passo do treinamento (NIELSEN, 2018). No geral, aumenta-se a quantidade de casos para a rede treinar para que ela atinja um aprendizado ótimo, mas existe a possibilidade de ela treinar tanto a ponto de degradar a sua capacidade de generalização da predição, o chamado ***overfitting*** (BROWNLEE, 2017). Já durante a predição, assume-se que a rede já está treinada o suficiente para executar o exato mesmo processo, porém sem o gabarito, e obter um resultado final condizente. O processo que é utilizado para treinar ou prever varia de acordo com cada tipo de rede neural.

2.1.2 LSTM

A LSTM (*Long Short-Term Memory*) é um tipo de rede neural recorrente, e com isso, é uma forma útil de se prever informações sequenciais, tais como clima, índice de uma bolsa de valores, recomendação de produtos, entre outras (BROWNLEE, 2017). No caso desta pesquisa, cada parâmetro musical, que será melhor explicado no Capítulo 3, é dado também de forma sequencial, o que adéqua este tipo de rede ao uso desejado.

Como LSTMs são recorrentes, neurônios de ativação de camadas anteriores à camada atual são usados na geração de uma saída, e não apenas os imediatamente antes. Podemos ver na Figura 6 os caminhos que vêm de unidades de processamento de cada entrada imediatamente anteriores, relativos à memória de curto prazo; e os que vêm de células anteriores mais distantes, relativos à memória de longo prazo.

Mas para que os pesos de suas camadas não cresçam indefinidamente ou assumam valores muito baixos, elas possuem unidades de memória, chamadas de **células**, ou **blocos**, conforme é exibido na Figura 7. Tais situações ocorrem em redes neurais recorrentes desenvolvidas anteriormente, o que explica o vasto uso de LSTMs na atualidade (BROWNLEE, 2017).

O caminho da memória de longo prazo pode ser modificado pelas operações de multiplicação e soma, mas ele não é diretamente modificado por pesos e vieses, o que garante que o gradiente não exploda ou desapareça. Já o caminho da memória de curto prazo é diretamente modificado pelos pesos e vieses.

O processo dentro de uma célula é dividido em passos, explicitados na Figura 7. São eles:

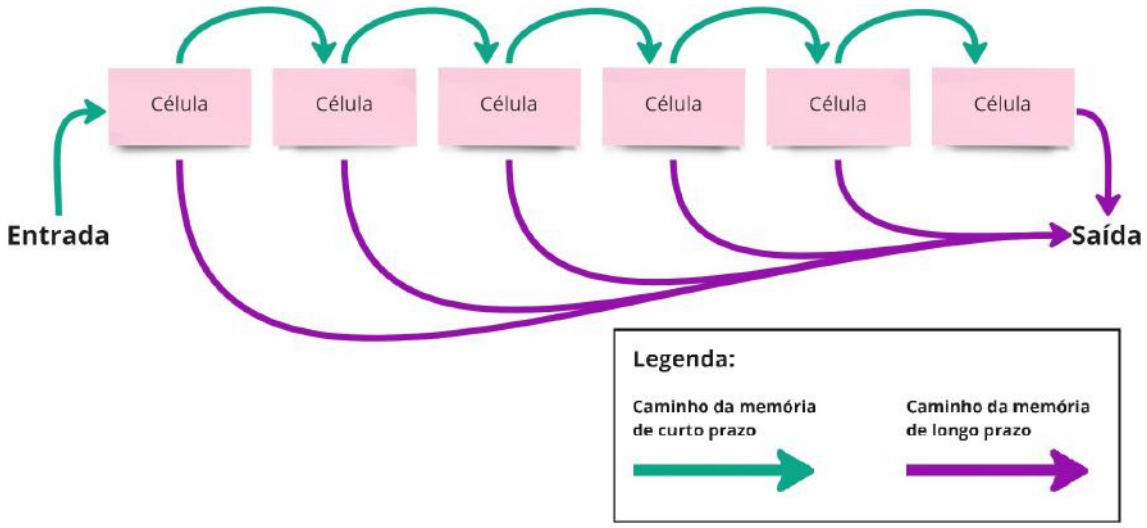

Figura 6 – Diagrama dos caminhos de uma rede LSTM.

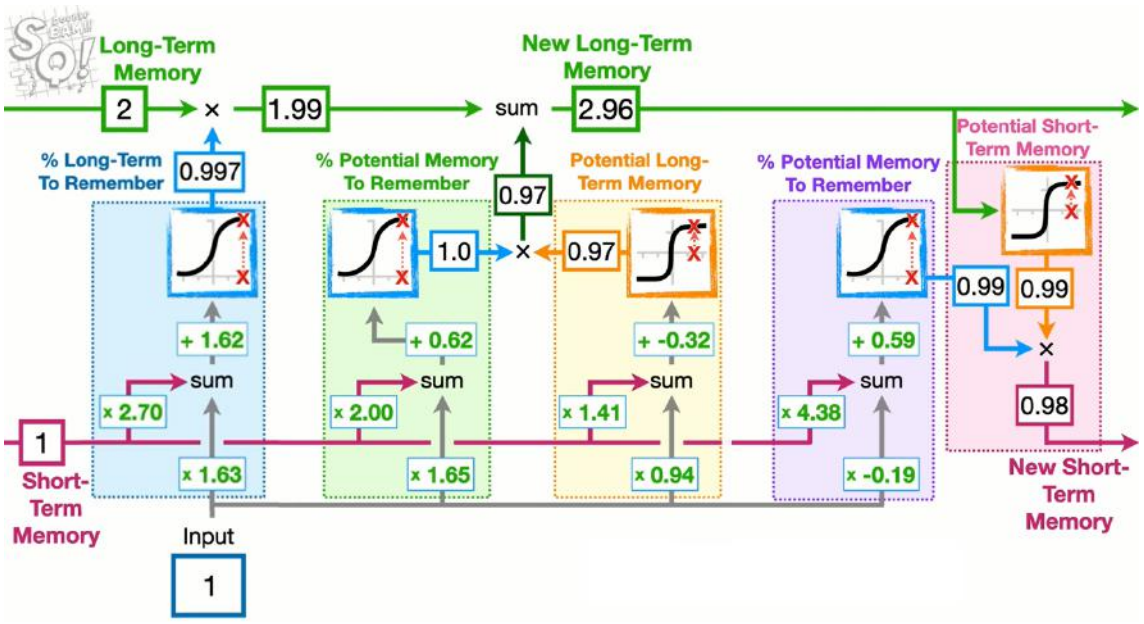

Figura 7 – Diagrama de uma célula ou bloco de memória de uma LSTM, com um exemplo prático. A memória de curto prazo é representada pela linha inferior cor de rosa que corta a figura horizontalmente. Já a memória de longo prazo é representada pela linha superior verde que corta a figura horizontalmente. - Fonte : https://youtu.be/YCzL96nL7j0. Acesso em 7/11/2022.

1. Soma-se o valor proveniente do caminho da memória de curto prazo e seu peso $(1 \times 2,70)$ com o valor da entrada e seu peso $(1 \times 1,63)$ com o viés $(1,62)$. Após isso, esse valor $((1\times2,70)+(1\times1,63)+1,62 = 5,95)$ é aplicado na função de ativação sigmoide. Essa função transforma qualquer entrada em um valor entre 0 e 1, o que podemos perceber como uma porcentagem. O valor resultante $(e^{5,95}/(e^{5,95} + 1) = 0,997)$ é multiplicado pelo valor do caminho da memória de longo prazo. Esse passo, então, determina a porcentagem dessa memória que será "lembrada".

2. Igualmente ao passo anterior, soma-se a memória de curto prazo $(1 \times 2,00)$ com a entrada $(1 \times 1,65)$ e o viés $(0,62)$, e aplica-se esse valor na função sigmoide. Novamente temos uma porcentagem $(1,0)$, mas dessa vez ela é multiplicada pelo resultado de também se somar a memória de curto prazo $(1 \times 1,41)$ com a entrada $(1\times0,94)$ e o viés $(-0,32)$, porém agora aplicando à função de ativação da tangente hiperbólica $(e^{2,03} - e^{-2,03}/(e^{2,03} + e^{-2,03}) = 0,97)$. Esta função transforma qualquer entrada em um valor entre -1 e 1. Assim, calculamos a porcentagem de uma nova memória, e o valor resultante é somado pelo valor do caminho da memória de longo prazo $(1,99 + 0,97 = 2,96)$. Esse passo, por sua vez, cria uma potencial memória de longo prazo e determina qual porcentagem dela será de fato passada à frente no processo.

3. Aplica-se o valor da memória de longo prazo $(2,96)$ à função de ativação da tangente hiperbólica, e assim como no passo anterior, utilizamos a soma da memória de curto prazo $(1 \times 4,38)$ com a entrada $(1 \times (-0,19))$ com o viés $(0,59)$, aplicada à função de ativação sigmoide para determinar a porcentagem dessa nova memória de curto prazo a ser passada em diante $(0,99)$. Esses dois resultados são multiplicados $(0,99\times0,99 = 0,98)$, e assim, esse passo atualiza a memória de curto prazo da rede.

Ao final do processo, o caminho da memória de curto prazo representa a predição no passo em questão.

A cada célula, a LSTM reutiliza os mesmos pesos e vieses para que ela possa lidar com sequências de dados de diferentes tamanhos. Entretanto, ao tentar executar uma predição e errar, todos os pesos e vieses são recalculados pela própria rede antes de se começar o processo novamente. O processo no treinamento onde todas as células são executadas é chamado de época, e a quantidade de épocas a ser executada é definida pelo programador ou programadora à seu gosto.

2.2 FASE DE TREINAMENTO

Ao dar o conjunto de entrada, como uma única lista com todas as informações desejadas, a uma rede neural na fase de treinamento, essa entrada passa por seus neurônios artificiais, e o peso de suas arestas é o que ajuda em sua classificação pela rede. Na

primeira tentativa, a rede utiliza um *batch* de um tamanho X, significando que ela considera os X primeiros elementos para prever o elemento X+1. Após validar se era o valor esperado, os elementos a serem considerados deslocam em uma posição, para utilizar novamente X elementos para prever X+2, e assim por diante. A rede provavelmente classificará a entrada inconsistentemente nas primeiras tentativas, pois sua função de custo não estará em seu mínimo. Atestamos que isso ocorre ao fazer a comparação com a resposta esperada do conjunto de entradas de teste. Buscamos minimizar, então, a função de custo, utilizando o vetor gradiente. Dessa forma, os pesos serão ajustados e novamente as entradas do conjunto de teste serão fornecidas à rede, e o processo recomeça. Ao final, os pesos da rede estão ajustados corretamente, a função de custo se encontra no mínimo possível, e a rede passa a classificar as entradas conforme o esperado. Assim, temos uma rede treinada e pronta para ser aplicada em processos reais.

2.3 FASE DE PREDIÇÃO

Uma vez treinada, a rede, que gerou um arquivo final de pesos, é replicada, mas dessa vez com o intuito de prever um elemento a cada separação do tamanho do *batch*, ou um "lote" de informações. Conforme esses pesos, ela é capaz de prever elemento por elemento, até gerar um resultado final. Espera-se que a saída da rede seja uma saída válida, pois a fase de treinamento foi longa o suficiente para se corrigir a grande parte das falhas de aprendizado.

Para maiores referências sobre Redes Neurais e LSTM, consultar (NIELSEN, 2018), (GOODFELLOW; BENGIO; COURVILLE, 2016) e (BROWNLEE, 2017).

3 A MÚSICA

O advento da tecnologia permitiu que a música pudesse ser representada computacionalmente de várias maneiras, em especial, se tratando da manipulação de cada parâmetro musical[1], como no escopo desta pesquisa. Neste capítulo, exporemos os conceitos básicos da música e as formas de representá-la para que ela possa alimentar uma rede neural do tipo LSTM.

3.1 CONCEITOS MUSICAIS FUNDAMENTAIS

De maneira simples, uma obra musical é uma combinação de sons e silêncios ritmicamente organizados. Ela pode ser entendida como uma linguagem, em especial quando a abordamos da forma computacional. Nas subseções seguintes, falaremos do som, do silêncio e do ritmo. Os parâmetros e outros termos musicais serão destacados em negrito.

3.1.1 Som

O som é a propagação de uma onda mecânica através de um meio condutor (ar, sólido, água etc.) percebida pelas membranas dos nossos tímpanos. A frequência e a amplitude do som, duas grandezas de natureza física que o caracterizam, recebem denominações diferentes na música: **altura** e **dinâmica**, respectivamente. Devemos ainda definir o conceito de nota, que é o resultado da produção sonora de um instrumento qualquer.

Sons podem ser simples ou complexos e de altura definida ou indefinida. Sons simples são formados por uma forma de onda simples, como uma senoide, por exemplo. Sons complexos, por sua vez, consistem em um amálgama de diversos outros sons simultaneamente. O som da tecla de um piano, por exemplo, é complexo, pois sua forma de onda pode ser explicada pela composição de diversas senoides (sons simples) de diferentes frequências e amplitudes[2]. Quando as frequências dessas formas de onda produzem uma progressão aritmética, cuja razão é igual ao valor do elemento inicial, têm-se uma série harmônica e o som resultante tem uma altura definida. Quando, por outro lado, essas formas de onda não produzem uma série harmônica, o som tem altura indefinida, como quando batemos em uma mesa. Desta forma, se tomarmos um som senoidal com frequência de 1 Hz e construirmos uma progressão aritmética infinita de razão 1, teremos uma série harmônica e o som simultâneo resultante terá, portanto, uma altura definida. As frequências que

[1] Tradicionalmente, parâmetro musical é "qualquer variável composicional, por exemplo, altura, duração, instrumentação, volume". No original, "any compositional variable, e.g. pitch, note duration, instrumentation, loudness. The term, borrowed from physics, became current in the early 1950s, when Boulez, Stockhausen, and others were submitting each parameter to serial control." (LATHAM, 2011).

[2] A teoria clássica do som, explicada por Helmholtz, pela série de Fourier (HELMHOLTZ, 1954).

formam um progressão geométrica de razão 2 (1, 2, 4, 8, 16,...), indicadas na Figura 8 pela cor rosa, formam, o que se chama na teoria musical de relações de **oitava** e, como veremos a seguir, são musicalmente denominadas pelo mesmo nome, sendo diferenciadas unicamente por um fator denominado **registro**.

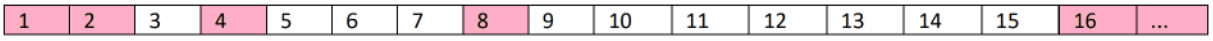

Figura 8 – Série harmônica, com as notas em rosa representando as oitavas.

Na música ocidental, as alturas são discretas, e identificadas por denominações latinas (Dó, Ré, Mi, Fá, Sol, Lá e Si) ou anglo-saxônicas (C, D, E, F, G, A e B) acrescidas de alterações ascendentes ou descendentes. Elas são distribuídas em ciclos de doze notas e se reproduzem em oitavas para cima e/ou para baixo (Figura 9). Os movimentos para cima de notas com o mesmo nome indicam o dobro da frequência e para baixo, a metade. Por exemplo, a nota central de um piano, conhecida como Dó central (ou Dó_3)[3], mede aproximadamente 261,6 Hz; o Dó imediatamente acima (Dó_4) tem o dobro da frequência, 523,2 Hz. Cada passo que uma nota dá para a nota seguinte é chamado de semitom, e dois semitons resultam em um tom.

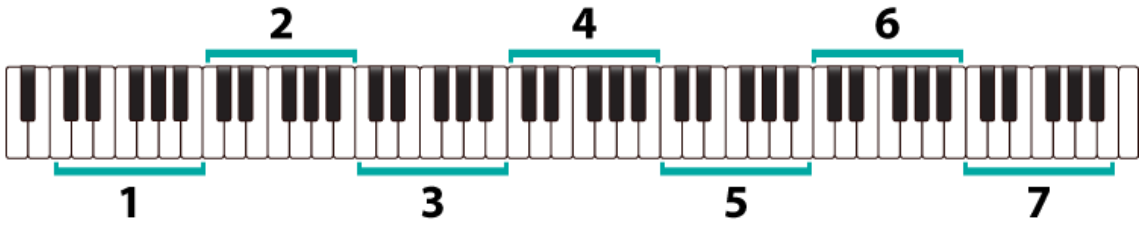

Figura 9 – Oitavas em um piano - Fonte : https://piano-music-theory.com/2016/05/28/the-piano-keyboard/. Acesso em 27/11/2022.

As 12 notas recebem os nomes de:

- Dó
- Dó♯ ou Ré♭
- Ré
- Ré♯ ou Mi♭
- Mi
- Fá
- Fá♯ ou Sol♭

[3] O índice 3 indica o parâmetro registro.

- Sol
- Sol♯ ou Lá♭
- Lá
- Lá♯ ou Si♭
- Si

Em algumas delas, temos ♯, chamado de sustenido, e ♭, chamado de bemol, representando acidentes, ou simplesmente símbolos, onde o ♯ aumenta um semitom da nota anterior, e o ♭ diminui. Notas que não possuem acidentes são chamadas de naturais. Na Figura 10, temos a representação dessas notas como teclas de um piano. As notas naturais são as teclas brancas, enquanto as notas com acidentes são as teclas pretas.

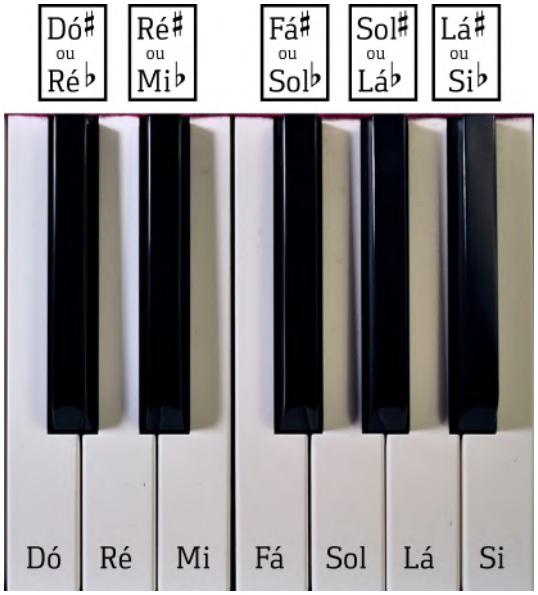

Figura 10 – Notas representadas por teclas de um piano.

Uma outra forma de representação das notas musicais é a **partitura**, notação gráfica contendo pautas de 5 linhas horizontais, onde as posições entre linhas ou sobre as linhas representam notas diferentes, como podemos ver na Figura 11. Elas contém **claves**, símbolos ao começo de cada pauta, simbolizando uma nota referencial, a partir da qual as outras são localizadas. Por exemplo, a clave de Sol (primeiro símbolo à esquerda na pauta superior da Figura 11) indica que a segunda linha dessa pauta é o Sol_3. Para indicar os acidentes, basta incluir os símbolos ♯ ou ♭ à frente da nota, como demonstrado na Figura 12.

Cada conjunto dessas 12 notas, como mencionamos anteriormente, são chamados de oitavas, e ao acabar uma oitava na nota Si, uma nova se inicia com a nota Dó mais aguda,

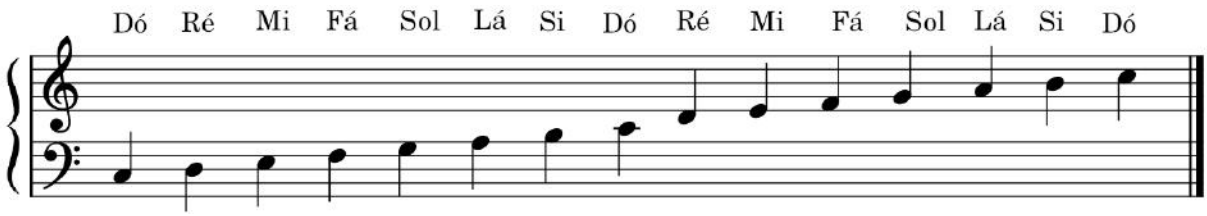

Figura 11 – Partitura de piano contendo notas nas duas pautas, a de cima representada pela clave de Sol e a de baixo representada clave de Fá.

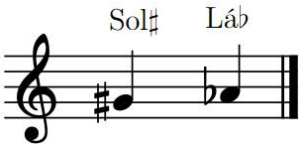

Figura 12 – Representação de Sol♯ e Lá♭ em uma pauta com clave de Sol.

ou de maior frequência, que a nota Dó da oitava anterior, ciclicamente. Como podemos ver na Figura 9, um piano tem aproximadamente 7 oitavas.

Cada som na música tem uma duração e uma dinâmica. Além disso, cada nota pode vir acompanhada ou não de outras notas simultâneas. Quando elas vêm acompanhadas produzem estruturas chamadas **acordes**. As durações mais comuns de uma nota são representadas graficamente pelas figuras chamadas de semibreve, mínima, semínima, colcheia, semicolcheia, fusa e semifusa, onde cada subsequente representa a metade da duração da anterior. Essas durações são representadas na pauta superior da Figura 13

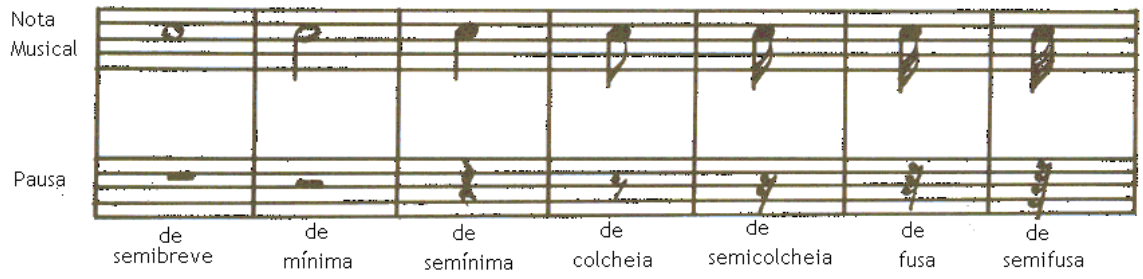

Figura 13 – Representação gráfica das figuras de duração de notas e pausas - Fonte: (KAROLYI, 1990)

3.1.2 Silêncio

O silêncio na música é chamado de pausa. Assim como as notas musicais, as pausas têm suas representações gráficas de duração diferentes variando de acordo com suas durações. São elas: pausa de semibreve, pausa de mínima, pausa de semínima, pausa de colcheia, pausa de semicolcheia, pausa de fusa, pausa de semifusa etc. Podemos ver suas representações gráficas na pauta inferior da Figura 13. A pausa de semibreve tem

a mesma duração de uma semibreve, a pausa de mínima tem a mesma duração de uma mínima, e assim por diante.

3.1.3 Tempo

O tempo de uma nota musical é definido por dois parâmetros:

- **duração**: o tempo que cada nota dura. Cada nota ou pausa explicitada na Figura 13 pode assumir também valores em segundos ou frações de segundos.
- **ponto de ataque**: também chamado de *offset*, o ponto no tempo no qual uma nota inicia.

Na Figura 14 podemos ver a representação em *Piano Roll*[4] de um trecho musical para exemplificar os elementos acima. O trecho se encontra dividido em 4 tempos, com as notas Dó_4, Mi_4, Sol_4, Dó_5 e Dó_4 ocorrendo em ordem cronológica. A nota Dó_4 tem duração de 1 tempo, sendo 1/4 do compasso, e começa a soar no *offset* 0. Assim, ao somar sua duração com o seu *offset*, sabemos que ela terminará de soar no *offset* 1. Como sabemos que a nota Mi_4 ocorre imediatamente depois de Dó_4, sem pausas, e que Dó_4 termina de soar em 1, sabemos assim que o *offset* de Mi_4 será 1. Somamos o *offset* 1, no final da duração de Dó_4, à duração de 1/2 tempo de Mi_4 para saber que ela termina de soar em 3/8 do compasso. Assim ocorre sucessivamente com as notas Sol_4, Dó_5 e o último Dó_4. A Figura 15 contém a mesma representação, porém em formato de partitura.

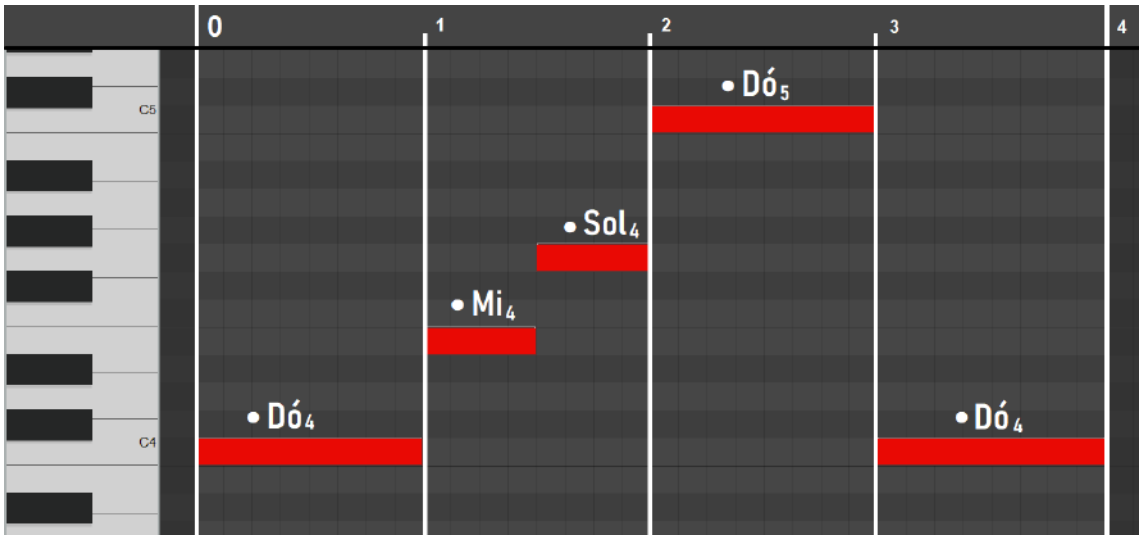

Figura 14 – Representação de um compasso no Piano Roll.

Em alguns casos, um evento musical pode ser dividido em mais partes do que normalmente seria. Este ocorrido é chamado de **quiáltera**. Por exemplo, podemos ter uma situação onde 3 semínimas encaixam no tempo de duração de 2 semínimas, gerando uma

[4] Um meio de representação de música, baseado em rolos de pianolas.

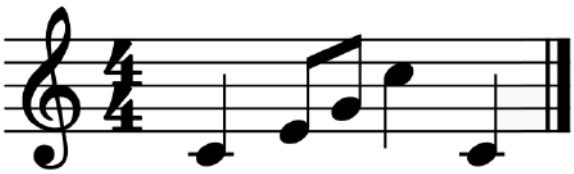

Figura 15 – Representação em partitura da peça da Figura 14

proporção de 3:2. Assim, o tempo de cada nota deixa de ser 1, e passa a ser 2/3, representado na Figura 16.

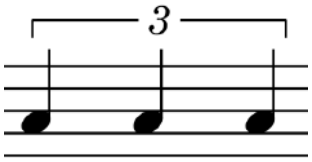

Figura 16 – Representação gráfica de uma quiáltera 3:2 de semínimas

Uma possível forma de se prolongar a duração de uma nota é por meio de uma **ligadura**, como vemos na Figura 17. Nessa representação, a nota é tocada apenas na primeira vez e a sua duração passa a ser a de ambas as notas somadas.

Figura 17 – Representação gráfica de uma ligadura entre duas semínimas

Outra forma é utilizando um ponto de aumento ao lado da nota, como na Figura 18. Ele aumenta a duração de uma nota em 50%.

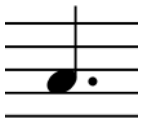

Figura 18 – Representação gráfica de semínima pontuada

3.1.4 Ritmo

O ritmo, um movimento marcado por uma sucessão regular de elementos de diferentes intensidades, é definido pela duração regular e pontos de ataque empregados em sons e

silêncios, em momentos estratégicos para que a música como um todo soe da maneira desejada.

O emprego de diferentes durações de elementos gera uma movimentação rítmica característica, que em certas situações, elas passam a representar gêneros musicais específicos, tais como rock, bossa nova, axé, entre outros.

3.1.5 Compasso

Um compasso musical é um segmento de tempo que corresponde a um número específico de batidas, determinado por uma fórmula. Seus limites gráficos são representados por barras verticais, como podemos ver na Figura 19. O compasso final é representado por duas barras, onde a última é mais grossa que a primeira. As batidas por compasso são definidas pelas fórmulas de compasso, como as exemplificadas na Figura 20. Por exemplo, em um compasso 2/4, haverá 2 batidas por compasso, onde cada batida é representada pela duração da semínima, representada pela fração 1/4 (lembremos que a semibreve vale 1), resultando na fórmula $2 * \frac{1}{4} = \frac{2}{4}$.

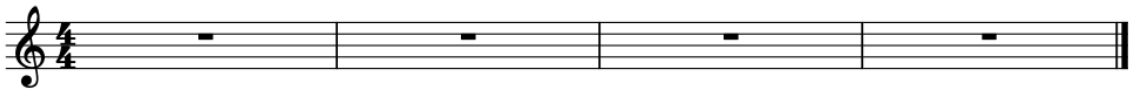

Figura 19 – Partitura com 4 compassos vazios.

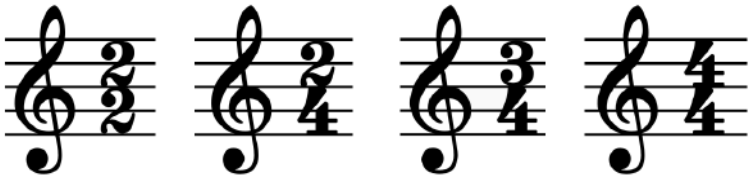

Figura 20 – Alguns exemplos de Fórmulas de Compasso.

Adicionalmente, devemos considerar que as obras musicais têm suas velocidades, denominadas **andamentos**[5], definidas em termos de batidas por minuto, popularmente conhecido como BPM. Essas batidas são aferidas por equipamentos denominados **metrônomos** (Figura 21). Assim, uma obra musical pode ter seu andamento indicado da seguinte maneira: ♩ = 60 BPM.

3.2 REPRESENTAÇÃO COMPUTACIONAL MUSICAL

A representação padrão da música que existe atualmente é por meio de partituras com todos os seus parâmetros, conforme explicado anteriormente. Para se trabalhar

[5] Andamento também pode ser chamado de "Tempo", mas não indica uma propriedade de passagem temporal propriamente, e sim algo que é relativo à velocidade da peça.

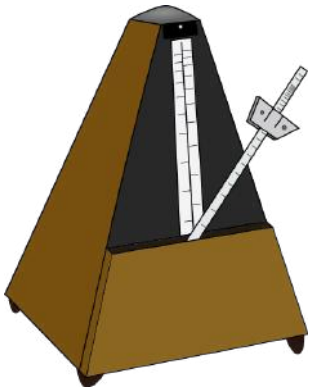

Figura 21 – Representação de um metrônomo.

com a música computacionalmente, partituras nem sempre atendem a demanda, ou às vezes ainda precisam ser expressas por meio de outros tipos de arquivo. Alguns formatos de arquivos conhecidos para essa representação são arquivos MIDI (*Musical Instrument Digital Interface*), de extensão mid, ou arquivos Music XML, de extensão mxl. Da mesma forma que uma partitura contém todas as informações listadas acima visualmente, esses arquivos as mantêm de forma digital.

Em certos tipos de trabalhos, esses formatos de representação musical se mostram desnecessariamente complexos, então é possível procurar formas de coletar apenas as informações necessárias a um tipo específico de tarefa. Para isso, há ferramentas disponíveis, como bibliotecas de composição e análise musical em linguagens de programação, tais como:

- **Music21**[6]: Desenvolvida por pesquisadores do MIT (*Massachusetts Institute of Technology*), Music21 é uma biblioteca de Python voltada para análise musical, que auxilia na extração de parâmetros de arquivos MIDI ou MusicXML. Ela foi desenvolvida de modo a trazer ao usuário informações mais próximas à realidade musical, diferente de outras opções que existem à disposição. Um claro exemplo disso é o parâmetro de duração, que não precisa ser calculado pelo usuário. Por ter seu foco em análise músical, dependendo da complexidade de uma peça, não é possível criar um arquivo de formato MIDI ou MusicXML fiel à composição. Um exemplo é a representação de quiálteras complexas. Caso sua representação de duração seja um número muito pequeno, em geral a partir de 6 casas decimais, a biblioteca as interpreta como não tendo nenhuma duração.

- **Mido**[7]: Em contrapartida à Music21, a biblioteca Mido, também de Python, tem um foco matemático. Certas informações são dadas ao usuário de forma crua, menos imediata, cabendo a ele fazer o tratamento dos parâmetros necessários para

[6] https://web.mit.edu/music21/
[7] https://mido.readthedocs.io/

sua tarefa. Tomando como exemplo o mesmo parâmetro citado acima, para calcular a duração de uma nota, o usuário precisa utilizar o tempo do evento de apertar uma nota, *note on*, e o tempo do evento de soltá-la, *note off*, para assim poder calcular sua duração.

- **Abjad**[8]: A biblioteca Abjad, de Python, foi criada com o intuito de auxiliar compositores a criar peças complexas, logo ela apresenta um viés menos analítico, e mais construtor.

Atualmente, todas elas apresentam algum ponto de fraqueza em relação a análises ou criação de composições, e, dependendo do escopo de um projeto, podem não satisfazer a tarefa sozinhas. No caso do programa de extração e montagem de peças, foi possível utilizar apenas a Music21 no processo.

[8] https://abjad.github.io/

4 O DESENVOLVIMENTO

Agora já temos a base de conhecimento necessária para desenvolvermos o planejado: uma rede que seja capaz de gerar uma nova sonata de Beethoven que, de um ponto de vista musical, seja condizente com suas próprias características.

O programa, intitulado bAIthoven, a ser utilizado para o desenvolvimento deste trabalho encontra-se público no GitHub [1].

4.1 O PROCESSO

Para entender o processo aplicado neste trabalho, podemos ver um diagrama na Figura 22. O fluxo se dá iniciando por um arquivo de música, em formato mxl. As informações musicais precisam ser extraídas dele para que seja possível utilizá-las no treinamento da rede. Esta extração foi feita através do programa de extração e montagem de peças, desenvolvido em Python (comunicação pessoal)[2], que gera um arquivo de texto com a informação musical a ser utilizada. Após isso, executamos uma tradução necessária para que a informação seja compreendida pela rede neural. Em seguida, podemos finalmente entrar na fase de treinamento ou predição. Após todo esse processo, é preciso fazer o caminho reverso, convertendo a saída da rede em um arquivo musical. Dessa forma, faz-se a tradução reversa condizente com o resultado do programa de montagem, para que ele seja capaz de fazer a remontagem da peça musical.

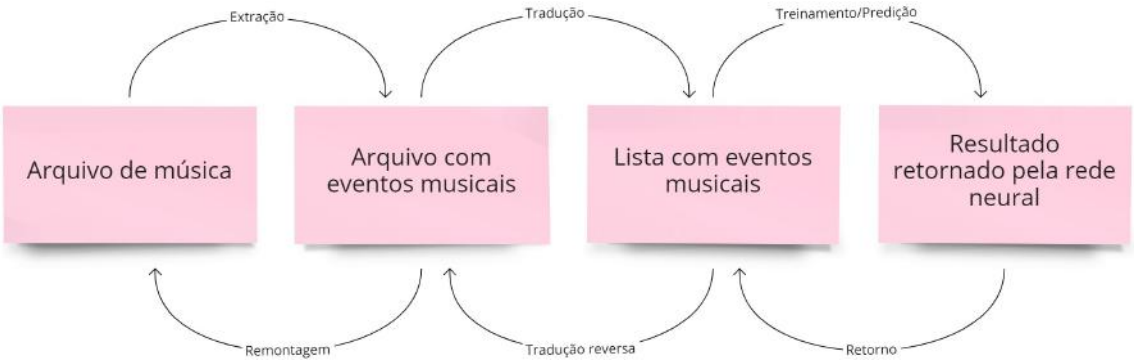

Figura 22 – Diagrama do processo

[1] https://github.com/siliconemonster/bAIthoven

[2] Conforme correios eletrônicos enviados pelo doutorando Pedro Proença em 22 de Novembro de 2022 e 19 de Julho de 2023.

4.2 EXPERIMENTO DE REFERÊNCIA INICIAL

Em (SKÚLI, 2017) é apresentado um exemplo de programa que utiliza uma rede LSTM, construída com a biblioteca Keras[3], para gerar música. A arquitetura da rede é exibida na Tabela 1. No treinamento, são usados 92 arquivos de música. Assim, entendemos o tratamento de notas, definição de camadas para as redes, definições de épocas, tamanhos de amostra, etc, do experimento de referência inicial para aplicar ao deste projeto.

Camadas	**Função de Perda**	**Tamanho do *batch***	**Épocas**	**Dicionário ordenado**
1. ReLU 2. Softmax	Categorical Crossentropy	128	200	Sim

Tabela 1 – Arquitetura do experimento referencial

Nesse experimento, foram utilizadas apenas as alturas de nota, aceitando as consequências de não incluir informações de tempo, ou concomitância de notas (acordes ou vozes) que seriam geradas. Isso implica em uma escolha do autor de como fazer a reconstrução das peças musicais e não da rede em si, respeitando apenas as alturas de notas geradas por ela. Com isso, as decisões de durações e *offsets* tornaram a peça repetitiva ao ouví-la.

4.3 SELEÇÃO DAS PEÇAS

Antes de iniciarmos a fase de treinamento, optamos por procurar sonatas já escritas em arquivos mxl e disponibilizadas gratuitamente na internet, no site MuseScore[4]. Não foi possível encontrar todas as 32 sonatas e seus movimentos, mas encontramos as seguintes 13 peças:

- Período Inicial:
  - Sonata 1, movimentos 1, 2, 3 e 4;
  - Sonata 8, movimentos 1, 2 e 3;
  - Sonata 9, movimentos 1, 2 e 3;
  - Sonata 10, movimentos 1, 2 e 3;
  - Sonata 12, movimentos 1, 2, 3 e 4;
  - Sonata 14, movimentos 1, 2 e 3;
  - Sonata 15, movimentos 1, 2, 3 e 4;

[3] Biblioteca em Python para a geração e utilização de redes neurais, disponível em: https://keras.io/

[4] Portal do programa de notação musical, que contém as peças disponibilizadas na seção de partituras: https://musescore.com/sheetmusic

- Período Intermediário:
  - Sonata 18, movimentos 1, 2, 3 e 4;
  - Sonata 21, movimentos 1, 2 e 3;
  - Sonata 23, movimentos 1, 2 e 3;
  - Sonata 24, movimentos 1 e 2;
- Período Tardio:
  - Sonata 29, movimentos 1, 2, 3 e 4;
  - Sonata 32, movimento 1 (a única incompleta, pois faltou o segundo movimento).

4.4 REPRESENTAÇÃO MUSICAL PARA UMA REDE NEURAL

Diferentemente de (SKÚLI, 2017), neste trabalho, optamos por não sacrificar possíveis resultados dado a seleção de parâmetros musicais específicos, e desta forma, escolhemos coletar todos os parâmetros que parecessem significativos para uma obra de Beethoven.

A primeira tentativa de gerar um modelo extraído de uma peça musical qualquer foi utilizando a biblioteca MIDO. Entretanto, as informações de tempo, duração, e *offset* de uma peça são necessariamente calculadas pelo programador, conforme visto anteriormente. Após algumas tentativas de acertar o cálculo, percebemos que havia uma alternativa: a biblioteca Music21, que, por ser uma ferramenta voltada para a análise musical, já possui um tratamento mais imediato.

Começamos então a segunda tentativa: extrair informações da peça utilizando a Music21. O processo de preparar a extração foi complexo e longo, já que a biblioteca funciona melhor para análise musical. Todos os parâmetros anteriormente citados no Capítulo 3 (**offset**, **tipo de evento**, **voz**, **duração**, **altura de nota** e **presença de ligadura**) foram corretamente extraídos, com a exceção das quiálteras. Quanto mais notas uma quiáltera contém, menor é a duração de cada nota da quiáltera. No caso da Sonata nº 8 de Beethoven, a Sonata Pathétique, podemos evidenciar este problema, como indicado na Figura 23. O motivo de isso ocorrer, é que o tempo de cada uma das notas se tornou tão pequeno que a biblioteca aproximou a 0, então essas notas deixariam de ser consideradas. Quiálteras são muito presentes nas sonatas de Beethoven, e se elas fossem desconsideradas, perderíamos parte de sua essência.

Assim, utilizamos o programa de extração e montagem de peças, que, apesar de também utilizar a biblioteca Music21 e também ter sido complexo e longo de preparar, trata de durações e *offsets* muito pequenos em forma de fração, por meio da função em Python "Fraction()" onde a aproximação não ocorre. Com ele foi possível extrair todas as in-

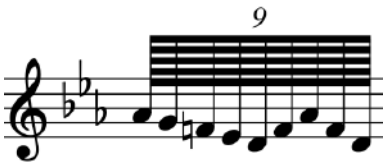

Figura 23 – Quiáltera de 9 quadrifusas (metade da duração de uma semifusa) na Sonata nº 8 de Beethoven, a Sonata Pathétique.

formações necessárias corretamente, a partir de um um arquivo teste de música .mxl, graficamente exibido pela partitura na Figura 24.

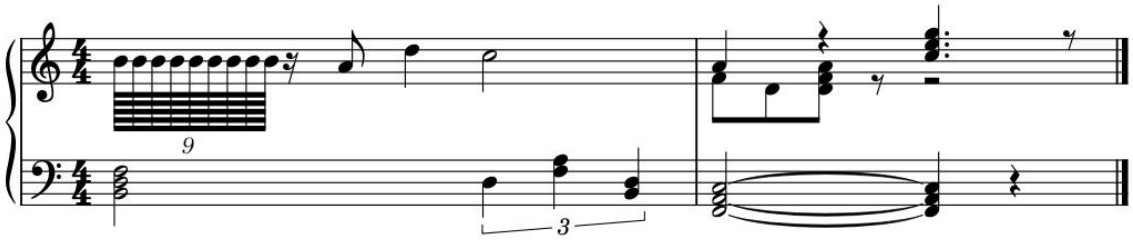

Figura 24 – Partitura de exemplo para a tradução das informações

A extração da Figura 24 resultou no conjunto do Código 1:

Código 1 – Extração da partitura exemplo

```
1. [Fraction(0, 1), 'Tempo', 120.0]
2. [Fraction(0, 1), 'Formula de Compasso', '4/4']
3. [Fraction(0, 1), 'Tuplet 9:8', 'Parte_1',
4.            [['Note', Fraction(1, 32), 71, None],
5.             ['Note', Fraction(1, 32), 71, None],
6.             ['Note', Fraction(1, 32), 71, None],
7.             ['Note', Fraction(1, 32), 71, None],
8.             ['Note', Fraction(1, 32), 71, None],
9.             ['Note', Fraction(1, 32), 71, None],
10.            ['Note', Fraction(1, 32), 71, None],
11.            ['Note', Fraction(1, 32), 71, None],
12.            ['Note', Fraction(1, 32), 71, None]]]
13. [Fraction(0, 1), 'Rest', 'Parte_2', Fraction(4, 1), 0, None]
14. [Fraction(0, 1), 'Chord', 'Parte_3', Fraction(2, 1), [47, 50, 53],
                                    None]
15. [Fraction(39, 160), 'Rest', 'Parte_1', Fraction(1, 4), 0, None]
16. [Fraction(79, 160), 'Note', 'Parte_1', Fraction(1, 2), 69, None]
17. [Fraction(159, 160), 'Note', 'Parte_1', Fraction(1, 1), 74, None]
18. [Fraction(319, 160), 'Note', 'Parte_1', Fraction(2, 1), 72, None]
19. [Fraction(2, 1), 'Tuplet 3:2', 'Parte_3',
20.           [['Note', Fraction(1, 1), 50, None],
21.            ['Chord', Fraction(1, 1), [53, 57], None],
22.            ['Chord', Fraction(1, 1), [47, 50], None]]]
```

```
23. [Fraction(639, 160), 'Tuplet 5:4', 'Parte_1', [['Rest', Fraction(1,
                                    128), 0, None]]]
24. [Fraction(4, 1), 'Note', 'Parte_1', Fraction(1, 1), 69, None]
25. [Fraction(4, 1), 'Note', 'Parte_2', Fraction(1, 2), 65, None]
26. [Fraction(4, 1), 'Chord', 'Parte_3', Fraction(2, 1), [41, 45, 48], '
                                    start']
27. [Fraction(9, 2), 'Note', 'Parte_2', Fraction(1, 2), 62, None]
28. [Fraction(5, 1), 'Rest', 'Parte_1', Fraction(1, 1), 0, None]
29. [Fraction(5, 1), 'Chord', 'Parte_2', Fraction(1, 2), [62, 65, 69],
                                    None]
30. [Fraction(11, 2), 'Rest', 'Parte_2', Fraction(1, 2), 0, None]
31. [Fraction(6, 1), 'Chord', 'Parte_1', Fraction(3, 2), [72, 76, 79],
                                    None]
32. [Fraction(6, 1), 'Rest', 'Parte_2', Fraction(2, 1), 0, None]
33. [Fraction(6, 1), 'Chord', 'Parte_3', Fraction(1, 1), [41, 45, 48], '
                                    stop']
34. [Fraction(7, 1), 'Rest', 'Parte_3', Fraction(1, 1), 0, None]
35. [Fraction(15, 2), 'Rest', 'Parte_1', Fraction(1, 2), 0, None]
```

Nele, encontramos informações entendidas como o cabeçalho de uma peça: O tempo, a tonalidade e a fórmula de compasso, nas linhas 1, 2 e 3, respectivamente. Essas três informações podem ou não aparecer, dependendo de como o arquivo da peça foi gerado. Caso apareçam, essas informações são dadas com o *offset* 0.0, pois são as primeiras passadas. Os eventos de cabeçalho, além de contarem com seus nomes e *offsets*, contam também com seus valores específicos, localizados na terceira posição.

Além de cabeçalho, essa representação contém cada evento de nota, pausa, acorde ou quiáltera, com seus respectivos atributos:

- O ***offset*** contém uma função de Python do tipo Fraction com a informação de tempo.
- O **nome do evento** descreve o tipo do evento (*note, chord, rest, tuplet*).
- A **voz** contém uma identificação numérica, onde cada uma delas recebe o nome de parte, como, por exemplo, a voz 1 é chamada de Parte_1.
- A **duração**, assim como o *offset*, contém uma função Fraction com a informação de tempo do evento.
- A **altura** contém um inteiro que representa uma nota, ou, no caso de acordes, uma lista de inteiros representando notas, onde essas notas são únicas. Por exemplo, notas Dó de oitavas diferentes são representadas por inteiros diferentes.
- A **ligadura** contém ou um indicador de início (start), de fim (stop) de uma ligadura, a ausência de uma ligadura (None), ou ainda um indicador de que a nota está entre ligaduras (continue).

Temos duas exceções na disposição da informação exibida acima. No caso de eventos do tipo acorde, já que suas notas ocorrem simultaneamente, no atributo de notas encontramos um conjunto com todas as notas do acorde. Já no caso de quiálteras, as informações de *offset*, nome do evento e voz são apresentadas normalmente, mas o quarto atributo é um conjunto para representar a informação das notas de uma quiáltera. Para cada uma delas, temos nome do evento, duração, altura e ligadura, como ocorre da linha 3 à linha 12.

As representações de cada peça musical foram salvas em respectivos arquivos de formato csv, e, a partir dele, traduzidas para uma única lista de listas na linguagem Python. Essa lista contém todas as peças anexadas em sequência na mesma estrutura, onde cada uma das informações de um evento, uma sublista da lista externa, tem seus tipos devidamente estabelecidos, para que seja possível implementar uma rede neural que as tomassem como entrada.

Entretanto, nessa representação encontramos imediatamente dois problemas: Os cabeçalhos (linhas 1 e 2) e os eventos (restante das linhas) não possuem os mesmos tipos de atributos, e certos eventos têm informações aninhadas. Uma rede neural espera um conjunto de informações de mesmo tipo e mesma quantidade de atributos. Dessa forma, seria necessário ou uma nova forma de extração dos dados de uma peça musical, ou uma manipulação dos dados já existentes. Como houve a dificuldade, explicitada anteriormente, de gerar um novo modo de extração, optamos pela manipulação da lista que temos em mãos neste ponto.

Para isso, geramos um script que faz toda a divisão de cada um desses eventos, seguindo passos na seguinte ordem:

1. Inicialmente, removemos as informações de cabeçalho (linhas 1 e 2), já que a rede precisará de eventos do mesmo tipo (neste caso, todas as informações de notas, e nada mais). Dessa forma o evento que começava na linha 3 no Código 1 passa a se situar na linha 1.

2. Então, separamos todas as quiálteras para cada um dos eventos listados nelas. Por exemplo, as linhas 3 a 12 no Código 1 indicando a primeira quiáltera, passam a ser da forma:

Código 2 – Exemplo de quiálteras separadas

```
1. [[0, 'Tuplet 9:8 - Note', 'Parte_1', Fraction(1, 32), 71, None],
2.  [0, 'Tuplet 9:8 - Note', 'Parte_1', Fraction(1, 32), 71, None],
3.  [0, 'Tuplet 9:8 - Note', 'Parte_1', Fraction(1, 32), 71, None],
4.  [0, 'Tuplet 9:8 - Note', 'Parte_1', Fraction(1, 32), 71, None],
5.  [0, 'Tuplet 9:8 - Note', 'Parte_1', Fraction(1, 32), 71, None],
6.  [0, 'Tuplet 9:8 - Note', 'Parte_1', Fraction(1, 32), 71, None],
7.  [0, 'Tuplet 9:8 - Note', 'Parte_1', Fraction(1, 32), 71, None],
8.  [0, 'Tuplet 9:8 - Note', 'Parte_1', Fraction(1, 32), 71, None],
9.  [0, 'Tuplet 9:8 - Note', 'Parte_1', Fraction(1, 32), 71, None],
```

```
...
```

3. Após isso, separamos todos os eventos de acordes para cada uma de suas notas, gerando novos eventos com cada nota. O acorde da linha 14 no Código 1 passa a ser representado pelas linhas 14, 15 e 16 no Código 3. É importante ressaltar que inclusive os acordes situados em tuplas são divididos nesta fase, como é o caso do acorde na quiáltera na linha 21 que passa a ser representado pelas linhas 23 e 24:

Código 3 – Exemplo de acordes separados

```
  ...
14. [0, 'Chord', 'Parte_3', 2, 47, None],
15. [0, 'Chord', 'Parte_3', 2, 50, None],
16. [0, 'Chord', 'Parte_3', 2, 53, None],
...
23. [2, 'Tuplet 3:2 - Chord =2', 'Parte_3', Fraction(1, 1), 53,
                                    None],
24. [2, 'Tuplet 3:2 - Chord =2', 'Parte_3', Fraction(1, 1), 57,
                                    None],
...
```

4. Na próxima fase, dividimos as informações de *offset* e de duração em numerador e denominador de fração, para que seja possível manusear essas informações mais facilmente:

Código 4 – Exemplo de frações separadas em numerador e denominador

```
1. [[0, 1, 'Tuplet 9:8 - Note', 'Parte_1', 1, 32, 71, None],
...
```

5. Em seguida, transformamos todos os "None" em string e achatamos a lista de listas, para que todas as informações de cada evento fiquem representadas em sequência, resultando em uma única lista:

Código 5 – Exemplo de parâmetros do tipo **None** transformados em *string*

```
1. [ 0, 1, 'Tuplet 9:8 - Note', 'Parte_1', 1, 32, 71, 'None', 0, 1,
...
```

Dessa maneira, temos a informação musical simplificada pronta para ser utilizada como entrada à rede neural, para que ela comece a fase de treinamento.

Ao fim deste processo, já com a separação feita, temos, então os seguintes 8 parâmetros e seus possíveis valores de domínio:

1°. Um inteiro positivo (podendo também assumir o valor 0), equivalente a um numerador de uma fração de tempo, relativo ao *offset* do evento. A cada entrada, o

resultado de sua divisão pelo parâmetro seguinte precisa necessariamente ser maior que o da entrada anterior.

2°. Um inteiro estritamente positivo, representando o denominador de uma fração de tempo, relativo ao *offset* do evento, diretamente ligado ao primeiro parâmetro.

3°. Uma string contendo o nome do tipo de evento. Pode conter exatamente as strings "Note" representando uma nota; "Rest" representando uma pausa; "Chord" representando um acorde; ou então conter alguma variação de "Tuplet", indicando uma quiáltera. No geral, as quiálteras conterão, neste parâmetro, a proporção de notas, como no caso da Figura 16, onde temos um "3:2". Além disso, ela contém a informação do tipo de sub-evento, podendo também ser "Note", "Rest" ou "Chord", onde ainda é dado o número de notas do acorde, como "=2" contém duas notas. Os possíveis valores, dadas as peças iniciais são "Chord", "Note", "Rest", "Tuplet 12:7 - Note", "Tuplet 12:7 - Rest", "Tuplet 3:2 - Chord =2", "Tuplet 3:2 - Note", "Tuplet 3:2 - Rest", "Tuplet 5:4 - Note", "Tuplet 5:4 - Rest", "Tuplet 6:5 - Note", "Tuplet 6:5 - Rest", "Tuplet 7:4 - Note" e "Tuplet 9:8 - Note".

4°. Uma string representando em qual voz o evento em questão se encontra. Os possíveis valores, dadas as peças iniciais são "Parte_1", "Parte_2", "Parte_3", "Parte_4", "Parte_5" e "Parte_6".

5°. Assim como o primeiro parâmetro, este assume um valor inteiro positivo (podendo também assumir o valor 0), equivalente a um numerador de uma fração de tempo, mas desta vez relativo à duração do evento em questão. Neste caso, a ordenação deste parâmetros em cada um dos eventos não é importante.

6°. E também assim como o segundo parâmetro, este assume um valor inteiro estritamente positivo, representando o denominador de uma fração de tempo, dessa vez relativo ao parâmetro anterior.

7°. Um inteiro entre 0 e 127, onde os valores de 1 a 127 representam as alturas de notas musicais, e o valor 0 representa uma pausa.

8°. Uma string representando a presença, ou não, de uma ligadura. No caso de no evento em questão não haver, é representado pela string "None"; no caso de uma ligadura se iniciar neste evento, este parâmetro contém a string "start", e no caso de uma ligadura acabar neste evento, uma string contendo "stop". Se algum evento anterior ao atual teve o atributo da ligadura com o valor "start", e após isso não houve evento com o valor "stop", o evento atual é marcado como "continue", representando que ele está entre uma ligadura.

No restante deste trabalho, nos referiremos ao conjunto de domínios definido acima como o **domínio dos oito parâmetros**.

4.5 FASE INICIAL DE TREINAMENTO

Com todas as peças já coletadas e o processo de conversão já estabelecido, foi possível iniciar o processo de treinamento da rede LSTM com algumas modificações em relação a (SKÚLI, 2017). O experimento original de (SKÚLI, 2017) possui um dicionário de termos, sendo cada um deles um valor inteiro, representando as alturas de notas, que foi ordenado para que durante as fases de treinamento e predição ele pudesse indicar sempre os mesmos elementos. Dessa forma, as modificações feitas em relação a (SKÚLI, 2017), que originalmente tinha a arquitetura definida conforme a Tabela 1, foram: como a lista de entrada contém strings, não foi possível gerar um dicionário em ordem crescente de inteiros representando cada uma dessas informações, então a ordenação foi removida; para experimento inicial, adicionamos uma camada de *ReLU* à arquitetura da rede e aumentamos o tamanho do *batch* para 512 pois há mais informações por evento no caso desta pesquisa (Tabela 2). A quantidade aumentada foi arbitrária, apenas com fins de experimentação.

Camadas	**Função de Perda**	**Tamanho do *batch***	**Épocas**	**Dicionário ordenado**
1. ReLU 2. ReLU 3. ReLU 4. Softmax	Categorical Crossentropy	512	200	Não

Tabela 2 – 1ª tentativa de arquitetura da rede

Inicialmente, o programa foi executado em um computador Intel i5 com Placa de Vídeo NVIDEA GForce GTX 1050(4GB) e 8GB de memória RAM, que foi capaz de converter todas as peças para a grande lista única, a ser utilizada como entrada para o treinamento. Ao iniciar a fase de treinamento, o computador não teve capacidade de prosseguir. Não foi possível gerar uma estimativa para a duração de uma época de treinamento.

Passamos a utilizar, então, um computador Intel Xeon series 5500, sem Placa de Vídeo e memória RAM de 8GB. Ele foi mais veloz do que o computador utilizado inicialmente na fase de conversão das peças, mas ao chegar na fase de treinamento, também não conseguiu estimar um tempo de duração para cada época, e eventualmente a execução foi interrompida. Para podermos testar se o problema seria o tipo de rede, ou a quantidade de arquivos de música, decidimos executar o programa de (SKÚLI, 2017). Como seus arquivos de música eram menores, foi possível executar a fase de treinamento, onde cada época ficou com a estimativa de aproximadamente 47 minutos.

Encontramos um novo servidor, Intel Xeon E5-2600, com Placa de Vídeo e memóriam RAM de 16GB. Executamos o experimento de (SKÚLI, 2017) para evidenciar que o servidor possuía mais poder computacional, e pudemos ver que suas épocas passaram a estimar 16 minutos aproximadamente até a sua conclusão. Assim, decidimos seguir com o projeto descrito neste trabalho. Novamente tentamos executá-lo para estimar o tempo de uma época, desta vez com sucesso. Com a quantidade inicial de peças, tínhamos aproximadamente 1.300.000 elementos na lista, e aproximadamente 5916 termos distintos para esses elementos. Com isso, o servidor estimou 35 horas para cada época, o que se tornaria inviável ao tentar rodar 200.

Assim, foi necessário definir formas de reduzir a quantidade de informações a serem utilizadas na fase de treinamento. Considerando as 13 sonatas, foram feitas as seguintes tentativas:

1. Utilizando todas as informações de todas as 13 sonatas;
2. Utilizando apenas o 7º parâmetro, a altura de notas, de cada evento em todas as 13 sonatas;
3. Utilizando uma amostra de cada uma das peças, onde foi feita uma extração com uma quantidade de eventos apenas do começo de cada uma delas, onde esse começo foi determinado por uma possível finalização da parte da música, sem fazer um corte súbito;
4. Utilizando apenas os movimentos 1 de cada uma das amostras geradas no item 3;
5. Utilizando peças do período inicial;
6. Utilizando peças do período intermediário;
7. Utilizando peças do período tardio;

assim como amostras completas de cada um destes períodos (respectivamente, tentativas 8, 9 e 10) e considerando apenas o movimento 1 (respectivamente 11, 12 e 13). Finalmente, tentativa 14, usando apenas os movimentos da sonata 8, como uma tentativa de mudar o objetivo para gerar um novo movimento de uma sonata já existente.

Na Tabela 3, podemos ver cada uma das abordagens anteriores com

- o tamanho de seus dicionários: cada termo diferente que aquela peça possui;
- a quantidade de itens: cada elemento utilizado na entrada da rede, seja ele uma altura de nota, um nome de evento, uma presença de ligadura, etc;
- a duração de uma época de treinamento.

	Abordagem	Tamanho do dicionário	Quantidade de itens	Duração de 1 época
0	*Classical-Piano-Composer*	326	45.972	00h 16min 04s
1	Quantidade original de peças	5.916	1.296.952	34h 54min 53s
2	Apenas alturas de notas	79	162.119	05h 56min 15s
3	Todas as peças reduzidas	724	286.296	08h 24 min 54s
4	Apenas movimentos 1 reduzidos	605	103.520	02h 53 min 02s
5	Apenas sonatas do período inicial	2.891	548.976	15h 34min 43s
6	Apenas sonatas do período intermediário	5.268	472.344	13h 31min 16s
7	Apenas sonatas do período tardio	3.315	275.632	10h 50min 31s
8	Apenas sonatas reduzidas do período inicial	577	149.384	04h 08min 38s
9	Apenas sonatas reduzidas do período intermediário	590	91.792	03h 17min 59s
10	Apenas sonatas reduzidas do período tardio	442	45.640	01h 14min 28s
11	Apenas movimentos 1 reduzidos do período inicial	473	54.640	01h 33min 15s
12	Apenas movimentos 1 reduzidos do período intermediário	498	28.968	01h 02min 35s
13	Apenas movimentos 1 reduzidos do período tardio	424	19.912	00h 41min 25s
14	Apenas 8ª sonata	1.809	80.360	02h 15min 44s

Tabela 3 – Comparação de tamanho, quantidade e duração de cada tipo abordagem.

Nesta tabela também vemos uma comparação com o Classic-Piano-Composer (SKÚLI, 2017), de índice 0.

Apesar de a melhor duração de 1 época ser a do caso 13, onde utilizamos apenas os movimentos 1 reduzidos do período tardio, essa abordagem só contaria com 2 peças, e dessa forma não geraria muita variação. Assim, a abordagem escolhida para dar prosseguimento foi a 12, utilizando os movimentos reduzidos do período inicial[5], pois não apenas nesse caso contamos com 7 peças, mas também a sonata 8, motivo de escolha do tema deste trabalho, está presente entre elas.

Uma vez elegida a 12ª abordagem, executamos o treinamento da rede no servidor de maior poder computacional. O planejamento inicial foi de rodar por 200 épocas com a arquitetura de 3 camadas *ReLU* e uma *Softmax*, função de perda *categorical crossentropy*, e o tamanho do *batch* de 512; mas após um pouco mais de 6 dias, o processo foi interrompido do lado do servidor, resultando em 99 épocas rodadas, onde a perda da 1ª época foi de 4.1227, enquanto a da época 99 foi de 0.7879. Ao final desta execução, tínhamos um

[5] Disponibilizados em formato mxl em https://github.com/siliconemonster/bAIthoven/tree/main/src/pieces.

arquivo gerado de pesos relativo a 99 épocas de treinamento, e assim, a possibilidade de prosseguir para um teste de predição.

4.6 FASE DE PREDIÇÃO

4.6.1 Tradução reversa

Ao fazer o treinamento e a predição de uma nova peça musical, a rede neural devolve uma lista achatada com todos os eventos em sequência, condizente com a que ela utilizou como treinamento. Dessa forma, é necessário traduzi-la de volta para o formato da tabela fornecida pelo programa de extração e montagem de peças, para ela ser transformada novamente um arquivo musical e ser validada. O passo a passo é o oposto do descrito na Seção 4.4.

Após executar esse tratamento na peça final que a rede neural gerou, onde não é necessário readicionar Tempo, já que não é preciso especificar, ela está pronta para ser convertida de volta em peça musical pelo programa de extração e montagem de peças.

4.6.2 Predição

Com o arquivo de pesos e com a tradução reversa prontos, começamos o processo de predição de uma nova música. Assim, a predição iniciada foi feita baseada em 99 épocas de treinamento. Obtivemos os seguintes 62 eventos, previstos pela rede:

Código 6 – Resultado do primeiro experimento

```
1. [[115, 1, 'Chord', 'Parte_6', 1, 2, 68, 959],
2.  [44, 1, 975, 'Parte_6', 1, 1, 59, 959],
3.  [40, 1, 449, 4721, 1, 1, 0, 959],
4.  [116, 1, 'Chord', 19199, 1, 2, 56, 959],
5.  [120, 1, 'Chord', 'Parte_6', 1, 2, 68, 959],
6.  [120, 1, 449, 'Parte_6', 1, 1, 68, 959],
7.  [48, 1, 449, 4721, 1, 2, 0, 959],
8.  [120, 1, 'Chord', 19199, 1, 2, 56, 959],
9.  [120, 1, 'Chord', 'Parte_6', 1, 2, 66, 959],
10. [44, 1, 975, 19199, 1, 1, 59, 959],
11. [40, 1, 'Chord', 19199, 3, 1, 56, 959],
12. [113, 1, 975, 19199, 1, 2, 42, 959],
13. [113, 1, 975, 'Parte_6', 1, 2, 73, 959],
14. [81, 1, 975, 'Parte_6', 1, 2, 73, 959],
15. [113, 2, 975, 4721, 1, 2, 71, 959],
16. [122, 1, 975, 4721, 1, 2, 76, 959],
17. [120, 1, 'Chord', 4721, 1, 2, 76, 959],
18. [125, 1, 'Chord', 'Parte_6', 1, 2, 0, 959],
19. [125, 1, 975, 19199, 1, 2, 46, 959],
20. [141, 1, 975, 429, 1, 2, 49, 959],
21. [149, 1, 'Parte_1', 'Tuplet 7:4 - Note', 1, 1, 'Note', 959],
```

```
22. [149, 1, 975, 'Parte_4', 1, 1, 44, 959],
23. [150, 1, 975, 4721, 1, 2, 75, 959],
24. [150, 1, 'Chord', 4721, 1, 2, 68, 959],
25. [150, 1, 'Chord', 4721, 1, 2, 68, 959],
26. [150, 1, 'Chord', 4721, 1, 2, 75, 959],
27. [150, 1, 'Chord', 4721, 1, 2, 68, 959],
28. [151, 1, 'Chord', 4721, 1, 2, 68, 959],
29. [151, 1, 'Chord', 4721, 1, 2, 70, 959],
30. [165, 1, 'Chord', 4721, 1, 2, 73, 959],
31. [165, 1, 'Chord', 'Parte_6', 1, 1, 0, 959],
32. [165, 1, 'Tuplet 3:2 - Rest', 19199, 1, 1, 44, 959],
33. [165, 1, 'Tuplet 3:2 - Rest', 429, 4, 1, 0, 959],
34. [193, 1, 'Tuplet 3:2 - Rest', 'Tuplet 7:4 - Note', 4, 1, 0, 959],
35. [193, 1, 'Tuplet 3:2 - Rest', 'Parte_4', 4, 1, 0, 959],
36. [206, 1, 'Chord', 4721, 1, 2, 61, 959],
37. [186, 1, 'Chord', 4721, 1, 2, 66, 959],
38. [182, 1, 'Chord', 4721, 1, 2, 68, 959],
39. [182, 1, 'Chord', 4721, 1, 2, 63, 959],
40. [182, 1, 'Chord', 4721, 1, 2, 68, 959],
41. [183, 1, 'Chord', 4721, 1, 2, 68, 959],
42. [183, 1, 975, 'Parte_6', 1, 1, 73, 959],
43. [183, 1, 'Parte_1', 19199, 1, 1, 42, 959],
44. [201, 1, 'Parte_1', 19199, 1, 1, 49, 959],
45. [201, 1, 'Tuplet 3:2 - Rest', 429, 1, 1, 0, 959],
46. [217, 1, 'Tuplet 3:2 - Rest', 'Tuplet 7:4 - Note', 4, 1, 0, 959],
47. [201, 1, 'Tuplet 3:2 - Rest', 'Parte_4', 4, 1, 0, 959],
48. [206, 1, 'Chord', 4721, 1, 2, 61, 959],
49. [206, 1, 'Chord', 4721, 1, 2, 66, 959],
50. [206, 1, 'Chord', 4721, 1, 2, 68, 959],
51. [214, 1, 'Chord', 4721, 1, 2, 63, 959],
52. [214, 1, 'Chord', 4721, 1, 2, 66, 959],
53. [214, 1, 'Chord', 4721, 1, 2, 68, 959],
54. [215, 1, 975, 'Parte_6', 1, 1, 73, 959],
55. [235, 1, 'Parte_1', 19199, 1, 1, 42, 959],
56. [217, 1, 'Parte_1', 19199, 1, 1, 49, 959],
57. [217, 1, 'Tuplet 3:2 - Rest', 429, 4, 1, 0, 959],
58. [217, 1, 'Tuplet 3:2 - Rest', 'Tuplet 7:4 - Note', 4, 1, 0, 959],
59. [217, 1, 'Tuplet 3:2 - Rest', 'Parte_4', 4, 1, 0, 959],
60. [218, 1, 'Chord', 4721, 1, 2, 61, 959],
61. [206, 1, 'Chord', 4721, 1, 2, 66, 959],
62. [214, 1, 'Chord', 4721, 1, 2, 68, 959]]
```

Não foi possível fazer a tradução reversa da peça, uma vez que os resultados não respeitaram o domínio dos oito parâmetros. Dessa forma, a melhor abordagem de análise seria observar cada um dos parâmetros, e entender os pontos fracos e fortes da rede:

1°. **Numerador do *offset***: Apesar de apenas assumir valores inteiros positivos, como

deveria, não foi capaz de estabelecer a ordem esperada, combinado ao segundo parâmetro, onde cada evento sucessor ocorre ou ao mesmo tempo ou depois. É possível enxergar alguma tentativa de manter a ordem, mas em muitos casos valores menores foram inseridos, como nas linhas 1, 2, 3, 4 e 5.

2°. **Denominador do *offset***: Assim como o caso anterior, assumiu apenas valores que poderia, mas em nenhum caso variou, com exceção da linha 15. Foi quase sempre 1. Não está necessariamente incorreto, se analisado sozinho, mas este valor deve se combinar ao anterior para estabelecer a ordem dos eventos.

3°. **Nome do evento**: Em alguns casos recebeu valores não aceitos, como números inteiros ou informações de voz, as partes, como nas linhas 2 e 55. Além disso, eventos do tipo "Note" ou "Rest" nem chegaram a ser gerados neste parâmetro, o que não é esperado, dado a análise da peça como um todo.

4°. **Voz**: Também recebeu valores não aceitos como inteiros ou nomes de eventos, como nas linhas 3 e 21. Porém algo que a rede foi capaz de reproduzir é que havia apenas 6 possíveis valores diferentes, já que nas peças fornecidas, havia no máximo 6 vozes.

5°. **Numerador da duração**: Absolutamente todos os valores gerados nesse campo foram valores válidos e fizeram sentido. Assumindo apenas valores de 1 a 4, não obtivemos durações inexistentes (0) e nem valores absurdamente grandes.

6°. **Denominador da duração**: Neste parâmetro, assim como no anterior, todos os valores foram válidos e fizeram sentido. Como eles representam uma fração, se combinados um com o outro, também geraram valores aceitáveis.

7°. **Altura de nota**: No caso deste parâmetro, possivelmente o mais importante de todos, já que determina a nota, houve apenas um erro: ele gerou um "Note" em um evento, na linha 21. Fora esse caso, todos os outros receberam valores válidos.

8°. **Ligadura**: Este parâmetro gerou apenas um valor para todos os eventos, e foi um valor inválido: um inteiro.

Além disso, analisando a peça como um todo, podemos ver que em 2/3 dos casos, a rede foi capaz de entender que quando temos uma quiáltera X:Y, ela deverá aparecer X vezes seguidas na listagem de eventos. O único caso onde houve erro, a primeira linha da quiáltera (32) aparenta desconexão com o restante das linhas (33, 34 e 35), já que é definida uma quiáltera de pausas, e a sua altura não indica uma pausa, que é definida como 0, mas sim uma nota Lá$\flat_2$, de altura definida como 44. Se tratando do parâmetro de altura, além do caso anterior, em alguns eventos, ele gerou um valor 0 para acordes, o que não faz sentido, pois um acorde não é feito de pausa. É válido ressaltar, entretanto,

que na maioria dos casos em que o nome do evento continha a pausa, a altura foi 0, como deveria ser.

Após essa predição, experimentamos gerar 100 eventos com o arquivo de pesos de 100 épocas, com perda de 0.7740. O resultado foi muito similar, não houve muitas mudanças em cada um dos parâmetros. Tentamos seguir com as predições e gerar saídas com pesos de várias épocas entre 100 e 200 para confirmar que se manteriam, e o mesmo aconteceu.

Para tentar melhorar os resultados, executamos as seguintes alterações:

1. Para a diminuição da complexidade das informações utilizadas como entrada à rede, experimentamos a remoção do parâmetro das ligaduras, já que, ao não retermos essa informação, a duração de duas notas ligadas ainda se manteria, com a diferença de que não ouviríamos continuidade entre elas.

2. Por meio do cálculo de menor multiplicador comum, foi possível ajustar os atributos que continham numeradores de forma a remover da lista todos os parâmetros que se tratavam de denominadores. Mantendo esses valores salvos em um dicionário, ao gerar uma peça nova, seria possível recuperar todos os valores.

3. O ajuste seguinte foi utilizar apenas os atributos de numerador do *offset*, numerador da duração e altura de nota, descartando elementos importantes para a reconstrução da peça, porém menos relevantes musicalmente: nome do evento e voz.

No geral o resultado gerado foi muito próximo às execuções anteriores, com exceção da 3ª alteração, que gerou apenas eventos repetidos, então foi necessário continuar explorando possíveis mudanças.

4.7 NOVAS TENTATIVAS

Supomos que o fato de a rede acertar a quantidade da variação de cada atributo, porém errar os valores aceitos predefinidos no domínio dos oito parâmetros poderia estar relacionada a não-ordenação do dicionário, uma vez que seria possível a rede treinar com uma ordenação e predizer com outra. Assim, foi necessário já passar todos os atributos como inteiros ao iniciar a fase de predição. A próxima tentativa, então, foi separar os dicionários por atributo, dessa vez na fase de tradução, pois assim, além de se tornar possível ordenar um segundo dicionário de inteiros ao iniciar a fase de predição, reduziria a quantidade de elementos diferentes e padronizaria a reconstrução da peça, resultando na nova arquitetura definida na Tabela 4. Como a remoção dos campos de denominadores poderia ser resgatada apenas ao terminar a predição, optamos por duas alternativas: uma contendo 6 atributos (numerador de *offset*, nome do evento, voz, numerador de duração, altura de nota, ligadura) e uma contendo 5 (apenas removendo as ligaduras).

Por sua vez, na nova tentativa com 6 atributos, cada época durou aproximadamente 1 hora e 20 minutos e a rede apenas gerou eventos repetidos. Porém, cada tentativa previa

Camadas	Função de Perda	Tamanho do *batch*	Épocas	Dicionário ordenado
1. ReLU 2. ReLU 3. ReLU 4. Softmax	Categorical Crossentropy	512	200	Sim

Tabela 4 – 2ª tentativa de arquitetura da rede

resultados que se encaixavam no domínio dos oito parâmetros. Dessa forma, foi possível fazer a tradução reversa, adicionando um cabeçalho genérico, e obter eventos aceitos pelo programa de extração e montagem de peças. Apesar de só haver um evento por predição, sempre repetindo, os resultados eram do tipo:

Código 7 – Resultado utilizando ordenação dos dicionários com 6 atributos

```
1.  [[Fraction(0, 1), 'Tempo', '120.0'],
2.  [Fraction(0, 1), 'Formula de Compasso', '4/4'],
3.  [Fraction(124, 1), 'Note', 'Parte_3', Fraction(4, 1), 60, None],
...
24. [Fraction(124, 1), 'Note', 'Parte_3', Fraction(4, 1), 60, None],
25. [Fraction(124, 1), 'Note', 'Parte_3', Fraction(4, 1), 60, None],
...
80. [Fraction(124, 1), 'Note', 'Parte_3', Fraction(4, 1), 60, None]]
```

Como decidimos que a remoção das ligaduras seria uma perda menor, optamos por experimentar da mesma maneira que a anterior, porém as removendo, agora restando 5 atributos. O resultado gerado foi do tipo:

Código 8 – Resultado utilizando ordenação dos dicionários com 5 atributos

```
1.  [[Fraction(0, 1), 'Tempo', '120.0'],
2.   [Fraction(0, 1), 'Formula de Compasso', '4/4'],
3.   [Fraction(81, 1), 'Note', 'Parte_3', Fraction(4, 1), 0, None],
4.   [Fraction(81, 1), 'Note', 'Parte_3', Fraction(1, 2), 51, None],
5.   [Fraction(81, 1), 'Note', 'Parte_5', Fraction(4, 1), 52, None],
...
8.   [Fraction(107, 1), 'Chord', 'Parte_4', Fraction(1, 2), [60, 64],
                                    None],
9.   [Fraction(107, 1), 'Chord', 'Parte_5', Fraction(1, 2), [51], None],
10.  [Fraction(215, 2), 'Note', 'Parte_3', Fraction(1, 2), 56, None],
...
13.  [Fraction(45, 2), 'Rest', 'Parte_1', Fraction(1, 2), 68, None],
14.  [Fraction(45, 2), 'Rest', 'Parte_4', Fraction(1, 2), 51, None],
15.  [Fraction(45, 2), 'Note', 'Parte_4', Fraction(1, 2), 56, None],
...
24.  [Fraction(80, 1), 'Rest', 'Parte_2', Fraction(4, 1), 0, None],
25.  [Fraction(76, 1), 'Note', 'Parte_3', Fraction(1, 2), 44, None],
```

```
26.  [Fraction(77, 1), 'Note', 'Parte_1', Fraction(1, 2), 56, None],
...
30.  [Fraction(45, 2), 'Chord', 'Parte_1', Fraction(1, 2), [63, 63, 0],
                                    None],
31.  [Fraction(45, 2), 'Chord', 'Parte_2', Fraction(1, 1), [60], None],
32.  [Fraction(45, 2), 'Chord', 'Parte_4', Fraction(1, 2), [32], None],
...
70.  [Fraction(205, 2), 'Chord', 'Parte_1', Fraction(1, 2), [0], None],
71.  [Fraction(205, 2), 'Note', 'Parte_4', Fraction(1, 2), 44, None],
72.  [Fraction(205, 2), 'Chord', 'Parte_4', Fraction(1, 2), [56, 56],
                                    None],
73. ...
78.  [Fraction(17, 1), 'Chord', 'Parte_4', Fraction(1, 2), [59, 59, 59,
                                    59, 59, 60], None],
...
101. [Fraction(239, 2), 'Note', 'Parte_3', Fraction(1, 2), 55, None],
102. [Fraction(17, 1), 'Tuplet 3:2', 'Parte_3',
        [['Note', Fraction(1, 2), 55, None], ['Note', Fraction(1, 2),
        60, None]]],
103.  [Fraction(17, 1), 'Tuplet 3:2', 'Parte_3',
        [['Rest', Fraction(1, 2), 59, None], ['Rest', Fraction(1, 2),
        59, None], ['Rest', Fraction(1, 2), 59, None], ['Rest',
        Fraction(1, 2), 59, None], ['Rest', Fraction(1, 2), 59, None],
        ['Rest', Fraction(1, 2), 59, None], ['Rest', Fraction(1, 2),
        59, None], ['Rest', Fraction(1, 2), 59, None], ['Rest',
        Fraction(1, 2), 59, None], ['Rest', Fraction(1, 2), 59, None]]
```

Assim, a rede foi capaz de gerar um resultado muito próximo de respeitar o domínio dos oito parâmetros. Após gerar saídas diferentes utilizando o mesmo peso decorrente dessa tentativa, pudemos evidenciar alguns pontos onde a rede cometeu erros:

- Ela não foi capaz de ordenar os *offsets* crescentemente, condizendo com a passagem do tempo. A única função real desse atributo foi a possibilidade de unir as notas de acordes ou as notas de quiálteras que aconteciam concomitantemente;
- Houve situações onde ele produziu um evento chamado 'Note' com altura de nota igual a 0, como na linha 3; um evento chamado 'Rest' com altura de nota diferente de 0, como na linha 13; e um evento chamado 'Chord' contendo 0 entre suas notas, como na linha 30.
- Em algumas situações o evento 'Chord' continha alturas de notas repetidas, como na linha 30;
- O evento 'Chord' continha notas únicas, como na linha 9;

- As quiálteras nunca foram geradas obedecendo a quantidade de notas em relação à proporção X:Y. Houve casos onde havia menos que X notas por quiáltera, como na linha 102; e houve casos onde havia mais que X notas, como na linha 103.

Como esses erros não variaram de execução para execução, foi possível tomar decisões de forma a ajustar a fase de tradução reversa para tratar esses tipos de erros gerados:

- Os *offsets* foram substituídos, sendo recalculados levando em consideração a ordem em que cada evento aparecia, a duração do evento anterior e a voz em que o evento ocorria.
- Os nomes de evento foram substituídos:
  - no caso de a altura de nota conter um 0, e o nome do evento ser 'Note', foi substituído por 'Rest'. Também seria possível manter o 'Note' no evento, e escolher um valor arbitrário entre 1 e 127, mas tomar essa decisão reduziria a autonomia da rede por interferência humana.
  - no caso de a altura de nota conter um valor diferente de 0, e o nome do evento ser 'Rest', foi substituído por 'Note'. Essa decisão é similar ao caso anterior: como a rede já foi capaz de prever a altura de nota entre 1 e 127, basta alterar o nome do evento para condizer com ela.
  - no caso de a altura de nota conter um único valor, e o nome do evento ser 'Chord', foi substituído por 'Rest' no caso de o valor ser 0, e substituído por 'Note' no caso de ser outros valores. A decisão é a mesma do caso onde há um 'Note' de altura de nota 0.
- Algumas alturas de notas dentro dos acordes foram eliminadas:
  - no caso de haver algum 0 entre as alturas de notas;
  - no caso de haver alturas de notas repetidas dentro do mesmo acorde.
- Quiálteras foram ajustadas para conter a quantidade certa de notas. No caso de quiálteras X:Y, caso ela possuísse menos de X notas, a última nota foi copiada até chegar na quantidade X. No caso de haver além, a quiáltera foi dividida em várias quiálteras.

Assim, o resultado anterior passou a ser do tipo:

Código 9 – Resultado **ajustado** utilizando ordenação dos dicionários com 6 atributos

```
1.  [[Fraction(0, 1), 'Tempo', '120.0'],
2.   [Fraction(0, 1), 'Formula de Compasso', '4/4'],
3.   [Fraction(81, 1), 'Rest', 'Parte_3', Fraction(4, 1), 0, None],
4.   [Fraction(81, 1), 'Note', 'Parte_3', Fraction(1, 2), 51, None],
```

```
5.   [Fraction(81, 1), 'Note', 'Parte_5', Fraction(4, 1), 52, None],
...
8.   [Fraction(107, 1), 'Chord', 'Parte_4', Fraction(1, 2), [60, 64],
                                    None],
9.   [Fraction(107, 1), 'Note', 'Parte_5', Fraction(1, 2), 51, None],
10.  [Fraction(215, 2), 'Note', 'Parte_3', Fraction(1, 2), 56, None],
...
13.  [Fraction(45, 2), 'Note', 'Parte_1', Fraction(1, 2), 68, None],
14.  [Fraction(45, 2), 'Note', 'Parte_4', Fraction(1, 2), 51, None],
15.  [Fraction(45, 2), 'Note', 'Parte_4', Fraction(1, 2), 56, None],
...
24.  [Fraction(80, 1), 'Rest', 'Parte_2', Fraction(4, 1), 0, None],
25.  [Fraction(76, 1), 'Note', 'Parte_3', Fraction(1, 2), 44, None],
26.  [Fraction(77, 1), 'Note', 'Parte_1', Fraction(1, 2), 56, None],
...
30.  [Fraction(45, 2), 'Note', 'Parte_1', Fraction(1, 2), 63, None],
31.  [Fraction(45, 2), 'Note', 'Parte_2', Fraction(1, 1), 60, None],
32.  [Fraction(45, 2), 'Note', 'Parte_4', Fraction(1, 2), 32, None],
...
70.  [Fraction(205, 2), 'Rest', 'Parte_1', Fraction(1, 2), 0, None],
71.  [Fraction(205, 2), 'Note', 'Parte_4', Fraction(1, 2), 44, None],
72.  [Fraction(205, 2), 'Note', 'Parte_4', Fraction(1, 2), 56, None],
73. ...
78.  [Fraction(17, 1), 'Chord', 'Parte_4', Fraction(1, 2), [59, 60],
                                    None],
...
101. [Fraction(239, 2), 'Note', 'Parte_3', Fraction(1, 2), 55, None],
102. [Fraction(17, 1), 'Tuplet 3:2', 'Parte_3',
        [['Note', Fraction(1, 2), 55, None], ['Note', Fraction(1, 2),
        60, None], ['Note', Fraction(1, 2), 60, None]]],
103.  [Fraction(17, 1), 'Tuplet 3:2', 'Parte_3',
        [['Rest', Fraction(1, 2), 59, None], ['Rest', Fraction(1, 2),
        59, None], ['Rest', Fraction(1, 2), 59, None]]],
104.  [Fraction(17, 1), 'Tuplet 3:2', 'Parte_3',
        [['Rest', Fraction(1, 2), 59, None], ['Rest', Fraction(1, 2),
        59, None], ['Rest', Fraction(1, 2), 59, None]]],
105.  [Fraction(17, 1), 'Tuplet 3:2', 'Parte_3',
        [['Rest', Fraction(1, 2), 59, None], ['Rest', Fraction(1, 2),
        59, None], ['Rest', Fraction(1, 2), 59, None]]],
106.  [Fraction(17, 1), 'Tuplet 3:2', 'Parte_3',
        [['Rest', Fraction(1, 2), 59, None], ['Rest', Fraction(1, 2),
        59, None], ['Rest', Fraction(1, 2), 59, None]]],
```

Com esses ajustes feitos, foram geradas 5 peças para serem analisadas do ponto de vista musical.

Também experimentamos executar a mesma tentativa de 5 atributos, porém aumen-

tando o tamanho do *batch* para 1024 e utilizando 250 épocas de treinamento, como na Tabela 5, e acrescentando mais duas camadas *ReLU* com 150 épocas, como na Tabela 6. O resultado obtido em ambos os casos foi semelhante à situação com tamanho do *batch* de 512 e 200 épocas.

Camadas	**Função de Perda**	**Tamanho do *batch***	**Épocas**	**Dicionário ordenado**
1. ReLU 2. ReLU 3. ReLU 4. Softmax	Categorical Crossentropy	1024	250	Não

Tabela 5 – 3ª tentativa de arquitetura da rede

Camadas	**Função de Perda**	**Tamanho do *batch***	**Épocas**	**Dicionário ordenado**
1. ReLU 2. ReLU 3. ReLU 4. ReLU 5. ReLU 6. Softmax	Categorical Crossentropy	128	150	Sim

Tabela 6 – 4ª tentativa de arquitetura da rede

Ao todo, foram geradas 11 peças[6]: 5 utilizando a arquitetura exposta na Tabela 4, 3 utilizando a da Tabela 5, e 3 utilizando a da Tabela 6. Para uma melhor visualização, na Tabela 7 adicionamos, além da arquitetura, a duração de cada peça e uma url direcionando para os arquivos de áudio de cada uma delas.

4.8 ANÁLISE DAS PEÇAS GERADAS

Durante o desenvolvimento, foi necessário determinar uma maneira de conversão e representação das peças musicais para um formato a ser utilizado de entrada para as fases de ftreinamento e predição da rede. Utilizamos um programa de extração e montagem de peças capaz de tanto coletar as informações da peça e disponibilizá-las em um formato de lista em Python, quanto pegar as informações da lista e trazê-las novamente para formato de peça musical.

Ao iniciar o treinamento, notamos que uma maior quantidade de sonatas, com a tentativa de utilizar todas as disponibilizadas, se mostrou inviável devido à sua complexidade de processamento: ao tentar utilizar todas, teríamos além de 7000 horas por treinamento.

[6] As 11 peças geradas podem ser encontradas em listagem, .mp3, .mxl e .pdf em https://github.com/siliconemonster/bAIthoven/tree/main/src/outputs/Final%20project%20results.

Peça	batch	Rede camadas	épocas	Duração em segundos	MP3
1A	512	3ReLU+1Softmax	200	21	link
1B	512	3ReLU+1Softmax	200	11	link
1C	512	3ReLU+1Softmax	200	21	link
1D	512	3ReLU+1Softmax	200	19	link
1E	512	3ReLU+1Softmax	200	19	link
2A	1024	3ReLU+1Softmax	250	35	link
2B	1024	3ReLU+1Softmax	250	39	link
2C	1024	3ReLU+1Softmax	250	7	link
3A	1024	5ReLU+1Softmax	250	27	link
3B	1024	5ReLU+1Softmax	250	23	link
3C	1024	5ReLU+1Softmax	250	25	link

Tabela 7 – Parâmetros sobre as peças geradas

Assim, reduzimos o escopo e coletamos apenas fragmentos dos primeiros movimentos das sonatas do primeiro período de Beethoven. Isso reduziu o tempo de treinamento a aproximadamente 200 horas, considerando em torno de 1 hora por época. Dessa forma, o experimento se tornou mais viável.

Entretanto, com essa redução de material, nenhuma das redes, com suas arquiteturas distintas, foi capaz de gerar uma saída passível de tradução imediata de volta a peça musical, devido a muitas inconsistências entre atributos, necessitando de um mínimo pré-processamento padronizado.

Ao gerar cada peça final, percebemos problemas do ponto de vista musical, com a ajuda de um especialista (comunicação pessoal)[7]. Na Figura 25, observaremos algumas características na Sonata 1 de Beethoven para embasar a análise o resultado da rede.

A primeira caixa azul (*a1*) indica uma estrutura que pode ser denominada de semifrase (compassos 1 e 2)[8], junto com uma repetição variada (*a1'*, nos compassos 3 e 4). Juntas, elas formam a primeira frase da peça. Já na segunda frase, notamos que os motivos[9] finais de cada uma dessas semifrases (caixas em vermelho) se repetem ao início da segunda semifrase.

O que é realizado na mão direita, na pauta superior, está no domínio da linha melódica, a dimensão horizontal da música. Já na mão esquerda do piano, na pauta inferior, observamos o acompanhamento harmônico, ou dimensão vertical. Vemos que esse acompanhamento é caracterizado por três fatores:

1. Regularidade rítmica;

[7] Conforme correio eletrônico enviado pelo compositor e professor Liduino Pitombeira em 13 de Agosto de 2023.

[8] O compasso incompleto inicial não é considerado neste caso.

[9] O motivo pode ser considerado como a subdivisão de uma semifrase (STEIN, 1979).

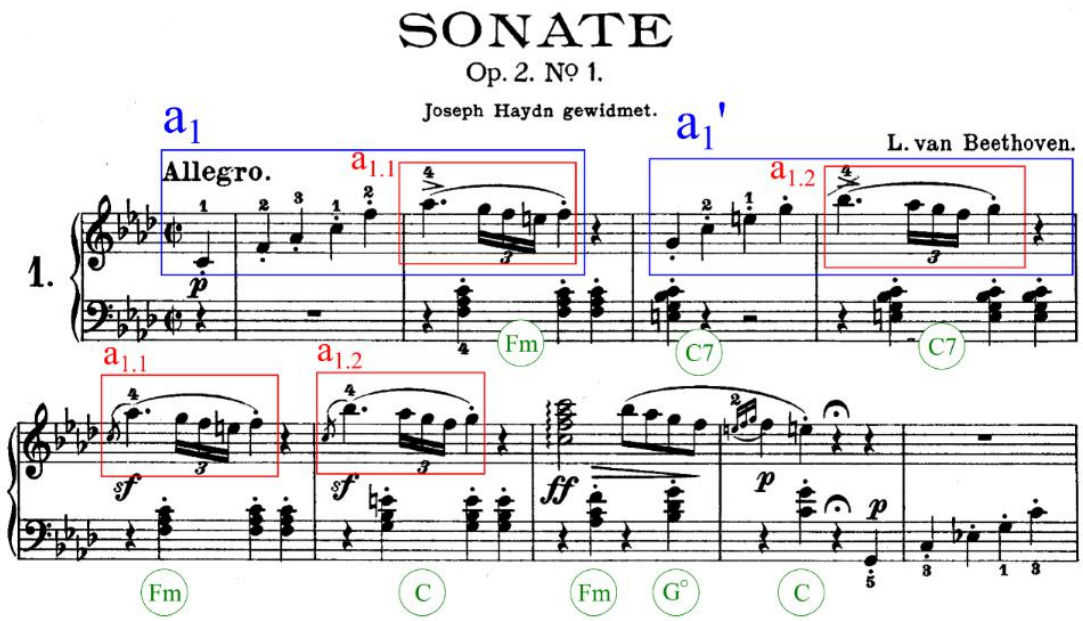

Figura 25 – Trecho inicial da Sonata Op.2 n.1 de Beethoven.

2. Uniformidade léxica[10] restrita a uma tríade maior, uma tríade menor, uma tríade diminuta e uma tétrade com sétima;

3. Economia harmônica: apenas quatro acordes são utilizados (Fm, C, C7, G°)[11], sendo um deles (C7) apenas uma expansão de uma tríade já presente.

Observamos que a melodia da mão direita concorda com a harmonia da mão esquerda: essa concordância se dá pela predominância das classes de notas[12] do acorde na melodia.

Uma vez entendidos esses conceitos, podemos analisar um fragmento da saída 1A, exibido na Figura 26. Destacamos 6 pontos que não apenas se afastam da música de Beethoven, mas também de toda a produção tonal de prática comum[13].

Em 1A-1 temos na pauta superior o que se poderia configurar como um motivo, potencial formador de uma linha melódica. Entretanto, esse motivo, além de estar desconectado do que vem no compasso posterior, pelo silêncio que a ele se segue, tem um âmbito não usual (16 semitons) que torna suas notas constituintes desconectadas em termos de fluxo melódico. A harmonia que acompanha esse motivo não se enquadra em um léxico harmônico tonal (acordes predominantemente maiores, menores e diminutos). Além disso, o núcleo dessa harmonia é um intervalo de segunda menor (Mi - Mi♭), uma dissonância que

[10] Neste caso, o termo léxico significa vocabulário harmônico.

[11] Representaremos notas com nomes latinos (Dó, Ré, Mi etc.) e acordes com nomes anglo-saxônicos (C, D, E etc.)

[12] Classes de notas são notas sem registro. Quando falamos de Dó4, temos um registro definido. Mas, quando mencionamos apenas Dó, trata-se de uma classe de notas, uma vez que não há um registro definido.

[13] Música produzida aproximadamente entre 1600 e 1900.

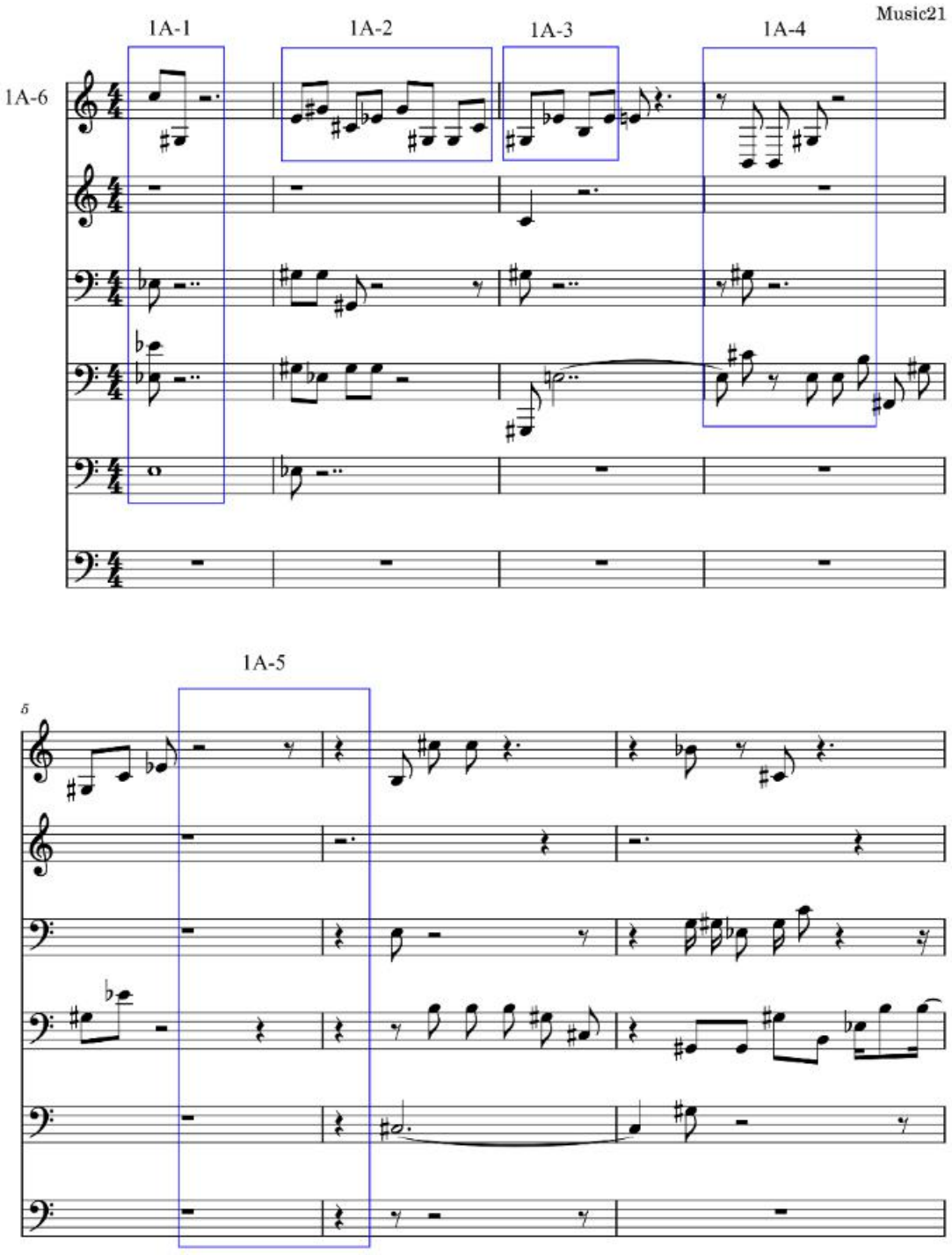

Figura 26 – Fragmento 1A gerado pela rede neural.

só acontece no sistema tonal em situações especiais, exigindo uma resolução para uma consonância.

O Fragmento 1A-2 apresenta o que poderia ser uma configuração tonal: uma tríade de C♯ menor com uma nota estranha (Ré♯). Porém essa nota estranha não apresenta as funções ornamentais usuais da música tonal (nota de passagem, bordadura etc.). Além disso, como essa nota estranha aparece no registro grave nas outras partes, o resultado sonoro não se assemelha a passagens tonais típicas desse período.

O Fragmento 1A-3 apresenta uma tríade de G♯ menor, mas a harmonia que dá sustentação é uma tríade aumentada, formada pelas notas Sol♯, Mi, Dó. Essa combinação simultânea não é típica do tonalismo.

O Fragmento 1A-4 apresenta, como total harmônico, uma tríade de C♯ menor com sétima menor, onde a sétima é a nota mais grave. O discurso interrompido, a desconexão dessa tríade com o que vem antes e depois, em termos de sintaxe tonal, torna o fragmento também inviável do ponto de vista musical.

O Fragmento 1A-5 mostra uma zona de silêncio que não é usual nas sonatas de Beethoven. O Fragmento 1A-6 rotulando toda a pauta superior mostra que os pequenos motivos não se conectam para formar uma linha melódica, o que é também não usual tanto em Beethoven, como em toda a produção tonal de prática comum.

Percebemos essas observações semelhantes em todos os fragmentos, de modo que nenhum dos resultados apresentou coerência léxica (de materiais) ou sintática (de conexão) tonal ou atonal: ainda são fragmentos soltos e desconexos, e não lembram Beethoven, que foi o repositório utilizado para treinamento da rede.

Além de 1A, podemos ver o exemplo 1B na Figura 27. Sua estrutura de alturas e de ritmo não apresenta similaridade com a produção tonal.

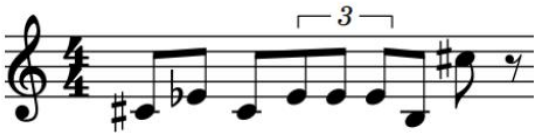

Figura 27 – Trecho inicial do Fragmento 1B gerado pela rede neural.

De maneira geral, não há uma linha melódica mínima, os acordes são raros e não tonais, e a quantidade de silêncio torna o resultado muito distante de música tonal e ainda mais de Beethoven. O Fragmento 1D ainda ensaia algo tonal, mas o discurso musical é muito soluçado e descontínuo e não se identifica, portanto, uma linha melódica.

5 CONCLUSÃO E TRABALHOS FUTUROS

O objetivo deste trabalho foi estudar a possibilidade de geração de novas peças de música, geração essa capaz de reproduzir o estilo de composição de Beethoven, utilizando o modelo de rede neural recorrente chamado de LSTM (*Long Short-Term Memory*). Como o estudo de composição musical por meio de aprendizado de máquina, em especial redes neurais, ainda é um campo em desenvolvimento, era esperado que as abordagens utilizadas neste estudo demandassem tempo, além de recursos computacionais, para se mostrarem bem sucedidas.

Das 32 sonatas para piano de Beethoven, utilizamos apenas 13 peças no experimento, devido à disponibilidade de seus arquivos, separadas por seus movimentos. Encontramos algumas dificuldades com essa coletânea: determinação de atributos de cada evento a ser utilizado, extração dos eventos para transformá-los em entrada para a rede, e complexidade computacional de treinamento com todo o conjunto de dados selecionado.

Optamos pela forma de representação que incluísse todas as informações necessárias para remontar a peça ao final do processo, e que abrangesse as características comuns em sonatas beethovenianas. Com isso foi necessário coletar informações além do mínimo necessário para representar uma peça musical simples. Seria possível, sim, utilizar esse mínimo necessário, mas, ao optar por essa representação, tanto parte das características de Beethoven se perderiam quanto a montagem da peça final não seria fiel à saída produzida pela rede, dado o modo de remontagem desenvolvido.

Entretanto, ao utilizar todas as informações disponíveis de todas as 13 peças, obtivemos um total de informações na ordem de milhão, e os recursos computacionais disponíveis dificultaram que uma investigação mais abrangente pudesse ser feita, como por exemplo, a busca de diferentes arquiteturas e hiperparâmetros no treinamento. Com isso, simplificamos as informações e a diminuímos as peças a serem utilizadas, o que impactou o resultado obtido. Se a rede apresentasse um processo de aprendizado mais rápido, talvez fosse possível usar mais sonatas e o resultado poderia ser melhor.

Logo, vemos que os desafios encontrados foram grandes e apontam para a necessidade de maior investigação, possivelmente na forma de representação da informação, da sua utilização, ou ainda em novas arquiteturas e hiperparâmetros, além de se fazer necessário dispor de um maior poder computacional.

Dessa forma, é possível, futuramente, complementar este experimento. Uma maneira é dispondo de maior poder computacional, para que seja possível aumentar a quantidade de camadas da rede e executar a fase de predição em tempo hábil para se tirar conclusões e seguir com os experimentos. Outra possibilidade é analisando o comportamento de outras redes neurais além da LSTM, uma vez que o modelo aqui apresentado talvez não seja capaz de gerar uma peça musical complexa coerente. Além disso, ainda seria possível

mudar a modelagem das informações musicais, talvez encontrando maneiras mais enxutas de representá-las, e tornar todo o processo de predição menos custoso, ainda que contenha informações de um número maior de sonatas.

REFERÊNCIAS

BRIOT, J.-P.; HADJERES, G.; PACHET, F.-D. **Deep learning techniques for music generation**. [S.l.]: Springer, 2020. v. 1.

BROWNLEE, J. **Long Short-Term Memory Networks With Python: Develop sequence prediction models with deep learning**. [S.l.: s.n.], 2017.

COPE, D. **Experiments in Musical Intelligence**. [S.l.]: A-R Editions, 1996.

CRESTEL, L.; ESLING, P. Live orchestral piano, a system for real-time orchestral music generation. 2016.

GERTSCH, N. **Beethoven's "Unfinished" (compositions)**. 2015. https://www.henle.de/blog/en/2015/02/02/beethoven%E2%80%99s-%E2%80%9Cunfinished%E2%80%9D-compositions/. Acesso em 26/04/2022.

GOODFELLOW, I. J.; BENGIO, Y.; COURVILLE, A. **Deep Learning**. Cambridge, MA, USA: MIT Press, 2016.

HELMHOLTZ, H. **On the Sensations of Tone**. [S.l.]: Dover Publications, Incorporated, 1954. (Dover Books on Music Series). ISBN 9780486607535.

IBM. **What is machine learning?**: Learn about the history of machine learning along with important definitions, applications, and concerns within businesses today. 2022. https://www.ibm.com/topics/machine-learning. Acesso em 09/07/2023.

KAROLYI, O. **Introdução à Música**. São Paulo: Martins fontes, 1990.

LATHAM, A. **The Oxford companion to music**. Oxford University Press, 2011. ISBN 9780199579037. Disponível em: https://books.google.com.br/books?id=ghY5AwEACAAJ.

MCCULLOCH, W.; PITTS, W. A logical calculus of ideas immanent in nervous activity. **Bulletin of Mathematical Biophysics**, 1943.

MUÑOZ-LAGO, P.; MÉNDEZ, G. **An approach to Beethoven's 10th Symphony**. 2020.

NIELSEN, M. A. **Neural Networks and Deep Learning**. [S.l.]: Determination Press, 2018.

SKÚLI, S. **How to Generate Music using a LSTM Neural Network in Keras**. 2017. https://towardsdatascience.com/how-to-generate-music-using-a-lstm-neural-network-in-keras-68786834d4c5. Acesso em 28/06/2022.

STEIN, L. **Structure & Style: The Study and Analysis of Musical Forms**. [S.l.]: Summy-Birchard Music, 1979. ISBN 9780874871647.

XENAKIS, I. **Musiques formelles: nouveaux principes formels de composition musicale**. Richard-Masse, 1963. (La revue musicale). Disponível em: https://books.google.com.br/books?id=KhWLrgEACAAJ.