# Proposta de módulo RNA para detecção de instrumentos em arquivos de áudio

**Lucas Lima**[1], **Alexandre de Oliveira Zamberlan**[1]

[1]Curso de Sistemas de Informação – Universidade Franciscana (UFN)
Santa Maria – RS – Brasil

lucas.lima@ufn.edu.br, alexz@ufn.edu.br

***Resumo.*** *Este estudo focou na análise de padrões em arquivos digitais de áudio, por meio de uma Rede Neural Artificial Perceptron, manipulando e analisando matrizes de sinais, com o intuito de reconhecer alguns instrumentos musicais presentes nesse áudio. A metodologia de pesquisa foi revisão bibliográfica amparada com estudo de caso. O estudo de caso foi o módulo RNA mais o processo de análise de arquivos de áudio. A metodologia de desenvolvimento de sistema é baseada no método ágil e customizável Scrum. Os resultados não foram os mais adequados, uma vez que os atributos de entrada (frequência e amplitude) ficaram comprometidos no processo de extração. Contudo, o trabalho apresenta algoritmos de uma RNA Perceptron (treino e teste) implementados em Python sem nenhuma biblioteca adicional, além do tratamento de arquivos de áudio para que atributos fossem identificados (reconhecidos).*

***Abstract.*** *This study analyzed patterns in digital audio files, using a Perceptron Artificial Neural Network, in an attempt to recognize some musical instruments present in this audio. The research methodology was a bibliographic review supported by a case study. The case study was the RNA module plus the process of analyzing audio files. The system development methodology is based on the agile and customizable Scrum method. The results were not the most adequate, since the input attributes (frequency and amplitude) were compromised in the extraction process. However, the study presents Perceptron RNA algorithms (training and testing) implemented in Python without any additional library, besides the treatment of audio files for attribute recognition.*

## 1. Introdução

A informatização está presente nos estúdios de gravação, com sistemas computacionais auxiliando músicos e produtores nos processos de gravação e melhorias das faixas sonoras gravadas. Dessa forma, a possibilidade de reconhecer, acrescentar e/ou retirar instrumentos do arquivo por meio de sua frequência e, gerar um novo arquivo de áudio, mantendo as configurações originais, pode ser interessante para aqueles que não têm acesso a mesas digitais que trabalham com canais de áudio.

A Inteligência Artificial possui inúmeras técnicas que auxiliam em processos de reconhecimento de padrões, como por exemplo, identificar objetos em imagens ou vídeos, ou elementos sonoros em arquivos de áudio. Uma subárea em destaque é a de Aprendizado de Máquina via Redes Neurais Artificiais, em que é possível ensinar um sistema

informatizado, por meio de treinamento repetitivo, que classifique ou categorize elementos diversos.

Este trabalho buscou projetar e desenvolver um módulo que pudesse identificar, em um arquivo de áudio do tipo WAV (*Waveform Audio File Format*), alguns instrumentos musicais, tendo como referência a análise de frequências sonoras em matrizes de uma dimensão (vetores). Para atingir o objetivo geral, assumiram-se alguns objetivos específicos: pesquisar o uso de arquivos de áudio na computação; identificar linguagens de programação e suas bibliotecas que processam arquivos de áudio; pesquisar e testar *frameworks* de programação para reconhecimento de padrões em arquivos de áudio; pesquisar interfaces de hardware para captar sistema de áudio; identificar trabalhos relacionados que analisaram e/ou processaram arquivos de áudio.

Uma justificativa para realização deste trabalho é a possibilidade de aplicar técnicas da área de Inteligência Artificial, com ferramentas consolidadas da Computação, agregando funcionalidades e, quem sabe, valor à indústria da música.

## 2. Revisão Bibliográfica

Esta seção contextualiza o trabalho nas áreas de Redes Neurais Artificiais (Reconhecimento de Padrões) e manipulação de arquivos de áudio, a fim de auxiliar o leitor na compreensão da proposta do trabalho.

### 2.1. Reconhecimento de Padrões

Reconhecimento de padrões é a classificação de objetos em categorias ou classes, com o intuito de facilitar o processo de identificação [Carneiro 2012]. Há várias aplicações de reconhecimento de padrões, dentre elas o reconhecimento de faces, identificação de íris e de digitais [Nogueira et al. 2006].

O Reconhecimento de Padrões possui três fases [Carneiro 2012]: extração de Características: mapeamento de todas características/atributos de um objeto a ser identificado; seleção das Características: as características extraídas do objeto são filtradas, para que as informações irrelevantes sejam removidas e somente as informações importantes sejam mantidas; construção de um Classificador: um classificador de características é criado com base nas características extraídas e filtradas do objeto, que serão os dados (e informações) que abastecerão o classificador.

### 2.2. Redes Neurais Artificiais como método para Reconhecimento de Padrões

Redes Neurais Artificiais (RNA) são sistemas distribuídos e paralelos de processamento de informações, capazes de armazenar conhecimento de forma experimental e disponibilizá-lo para uso, assemelhando-se as redes neurais biológicas do sistema nervoso humano. Tem por objetivo modelar e simular o cérebro humano por meio da aquisição de conhecimento pelo aprendizado (treinamento) e armazenamento do conhecimento pelas conexões presentes entre neurônios chamadas sinapses [Russel et al. 2013, Lima et al. 2009, Benite 2003]. A partir disso, quando uma Rede Neural treina, logo aprende, assim surge o conceito de Aprendizado de Máquina (*Machine Learning*).

As Redes Neurais possuem componentes fundamentais para o seu funcionamento [Russel et al. 2013, Santos et al. 2018]:

- Entradas: que podem ser valores fixos, ou valores definidos através de cálculos;
- Pesos: que são a intensidade da sinapse (importância de cada entrada), ou seja, a intensidade com que os sinais são transmitidos de um neurônio para o outro. Sendo assim, sinapse é a transmissão de sinais entre neurônios.

RNA possui também o treinamento, que consiste no ajuste/variação dos valores de entrada e dos seus respectivos pesos (teoria defendida e comprovada por Donald Hebb em 1949, que originou a Regra de Hebb, utilizada até os dias atuais em diversos algoritmos de treinamento de RNA [Santos et al. 2018]). Por meio do treinamento, as Redes Neurais chegam ao aprendizado, que é a repetição incessante do treinamento (calibragem dos pesos) até que elas possam fazer o reconhecimento de algo [Benite 2003].

As Redes Neurais possuem várias aplicações, dentre elas: Reconhecimento de Padrões em Economia e Negócios, Sistemas de Energia, Análise de Movimentos e Processamento e Transmissão de Sinais [Benite 2003]. Elas podem ser classificadas em três tipos [Russel et al. 2013, Benite 2003, Santos et al. 2018]: Quanto à Supervisão no Treinamento: Supervisionado, que possuem pares de treinamento, ou seja, vetores de entrada e vetores de saída calculados em somatório; Treinamento não Supervisionado que possui apenas vetores de entrada e desses são extraídas características; Quanto a Alimentação: *Feedforward* (Redes Diretas), redes neurais em que seus grafos não possuem ciclos e; *Feedback* (Redes com Ciclos ou com Realimentação), redes neurais em que seus grafos possuem pelo menos um ciclo; Quanto a Quantidade de Camadas: Redes de Camada Única, que possuem somente um nó entre entrada e saída e, Redes de Múltiplas Camadas, que possui mais de um neurônio entre entrada e saída.

A Função de Ativação consiste na soma ponderada dos sinais de entrada, em que a partir da soma será determinada a sua saída. Na maioria das vezes esta função é não-linear [Russel et al. 2013, Benite 2003]. A Figura 1 apresenta um código Python para um neurônio com dois vetores (*entradas* e *pesos*). O vetor *entradas* possui 3 valores inteiros representando 3 atributos quaisquer de uma RNA. O vetor *pesos* possui os 3 respectivos pesos para cada entrada existente no vetor *entradas*. O método *funcao_soma()* recebe os dois vetores e retorna o somatório. E o método *funcao_ativacao_step()* recebe o resultado do somatório e devolve 1 ou 0, ativando ou não o neurônio.

```python
entradas = [1  , 7  , 5]
pesos    = [0.8, 0.1, 0]

def funcao_soma(e, p):
    s = 0
    for i in range(len(e)):
        s += e[i] * p[i]
    return s

def funcao_ativacao_step(soma):
    if soma >= 1:
        return 1
    return 0

somatorio = funcao_soma(entradas, pesos)
ativacao = funcao_ativacao_step(somatorio)
print("Somatório: " ,somatorio)
print("Resultado ativação: ", ativacao)
```

**Figura 1. Código Python exemplificando as funções somatório e ativação (*step*).**

### 2.3. Padrões de arquivos de áudio

MIDI (*Musical Instrument Digital Interface*) é um protocolo que permite a comunicação entre instrumentos musicais e hardwares [Carneiro 2012]. Ele não possui dados de áudio,

somente instruções que serão lidas por meio de um sintetizador, podendo o mesmo ser hardware ou software, para serem executadas. O MIDI realiza um controle de dados, transmitindo as instruções em forma de pequenas mensagens, ou seja, se forem transmitidas várias mensagens de forma sequencial e, todas elas conterem notas, ao final da transmissão o sintetizador executará as instruções na sequência em que as mensagens foram recebidas e obterá como resultado uma melodia [Shinohara 2018]. A transmissão dessas mensagens ocorre através de canais MIDI, que somente transmitirão as mesmas se o sequenciador (ferramenta utilizada para fazer a gravação, edição, envio e reprodução dos dados MIDI) e o controlador (utilizado para enviar as instruções para o software que está sendo usado), estiverem configurados para utilizarem o mesmo canal. O arquivo MIDI ou *Standard MIDI Files* (Arquivos MIDI Padrão) tem como única função armazenar os dados MIDI, ou seja, as mensagens MIDI com suas respectivas informações. Esses arquivos são bastante utilizados em apresentações musicais e em estúdios [Carneiro 2012].

De acordo com [Canaltech 2020], arquivos WAV é um formato-padrão de arquivo de áudio pertencentes à Microsoft e à IBM para armazenamento de áudio em computadores pessoais. Um arquivo WAV pode ser compactado, via modulação de pulsos PCM (Pulse-Code Modulation). Novamente, conforme em [Canaltech 2020], PCM usa um método de armazenamento de áudio não-comprimido, para não ocorrer perda. Na prática, o formato WAV é de maior qualidade e pode ser editado e manipulado com relativa facilidade por meio de softwares.

### 2.4. Tratamento e análise de áudio

O tratamento de arquivos de áudio consiste em retirar fragmentos desnecessários para a análise do arquivo, ou seja, remover fragmentos que possam comprometer o resultado da análise. Assim, as informações relevantes ao processo de análise são preservadas [Carneiro 2012]. O tratamento de arquivos de áudio também pode ser caracterizado como a extração de parâmetros, ou as características do arquivo para posterior análise [Muniz 2009]. A análise de arquivos de áudio dá-se por meio de uma comparação entre as características extraídas de um arquivo de áudio com modelos de arquivos base ou modelos de referência [Muniz 2009]. A captura de arquivos de áudio ocorre por meio de sinais de áudio, extraídos dos arquivos por um software de análise. Após a captura vem o tratamento dos arquivos, em que dados são extraídos dos sinais de áudio capturados. Por fim, na análise, ocorre a categorização dos dados extraídos, síntese e comparação dos mesmos com modelos de arquivos base ou modelos de referência [Muniz 2009, Carneiro 2012].

### 2.5. Trabalhos relacionados

Nesta seção, há 3 trabalhos que realizaram pesquisa e utilizaram técnicas e ferramentas para análise de áudios.

O sistema proposto por [Shinohara 2018], utiliza o Sistema Classificador de Textura Musicais (SCTM), que faz uso de quatro conjuntos de características: características projetadas à mão; espectrogramas em escala Mel[1]; projeções aleatórias obtidas de espectrogramas[2] em escala Mel; características obtidas de uma rede de Autoencoder[3].

[1]Escala logarítmica perceptual com a finalidade de alinhar os tons de frequência, baseando-se 40 decibéis acima do limite de percepção humana, que está definido em 1000 Hz [Logan et al. 2000]

[2]Dados e informações de um sinal de áudio, sob análise tempo-frequência [Costa et al. 2013]

[3]Tipo de rede neural treinada para que os vetores de entrada e saída possuam as mesmas dimensões

No trabalho de [Carneiro 2012] - ”Desenvolvimento de um sistema de reconhecimento automático de músicas - SRAM”, na fase de extração de características, o processo é realizado em duas etapas. Na primeira etapa, é desenvolvido um Modelo Universal de Músicas (MUM), baseado na estimação de misturas Gaussianas. Na segunda etapa, faz-se uma estimação de símbolos em cada faixa do Modelo Universal de Músicas, identificando nas características de cada faixa o componente gaussiano, definindo uma sequência de símbolos através dos índices dos componentes do Modelo Universal de Músicas. A próxima fase é a classificação, em que é utilizado a modelagem do tipo $n - grama$ para a geração do MUM e os modelos específicos de cada artista, testando através do método de detecção para identificar o artista em cada música da base de amostras de teste. Finaliza-se o processo de decisão, em que o reconhecimento é tratado como um problema de detecção, tendo como opções de decisão duas hipóteses específicas: $H0$, caso o modelo produziu a sequência de símbolos desejada, ou seja, os artistas da faixa teste e de treinamento são iguais e, $H1$, caso os artistas sejam diferentes.

O projeto ”Reconhecimento de Ritmo Musical por Análise de Sinais de Áudio e Amplitude e Frequência”[Almeida 2009] foi realizado em três fases. A primeira é a fase de pré-processamento, em que foram extraídas informações dos arquivos de áudio por meio de técnicas de análise de amplitude e frequência modular. A segunda fase foi seleção de características, em que uma base de conhecimento foi gerada para extrair características dos estilos musicais. Por fim, a classificação, onde os dados das bases de conhecimento e de treinamento são comparadas e, a partir da comparação realizada, os resultados para a classificação foram gerados, ou seja, se os estilos musicais das músicas foram reconhecidos ou não.

Os trabalhos relacionados apresentam foco no reconhecimento de padrões, porém cada um adota técnicas e ferramentas distintas para a manipulação dos arquivos e no processo de reconhecimento. O trabalho de [Almeida 2009] aproxima-se mais desta proposta, pois além das técnicas de reconhecimento de padrões, ele utiliza também a análise de frequência de sinais de áudio, que é a forma de análise considerada ideal quando se deseja trabalhar com instrumentos musicais em arquivos de áudio. Dessa forma, [Almeida 2009] apresenta resultados e metodologia que podem contribuir na realização deste trabalho.

## 2.6. Tecnologias para Reconhecimento de Padrões em arquivos de áudio

Há algumas tecnologias que dão suporte à análise de arquivos digitais, como por exemplo:

1) Linguagem Python: linguagem de alto nível, orientada a objetos e com uma coleção de bibliotecas para os mais diversos fins [Borges 2014];

2) Biblioteca *NumPy*: é específica para a linguagem Python que fornece suporte para matrizes multi-dimensionais muito grandes [Comunidade de Desenvolvedores 2020];

3) Biblioteca Tensorflow: é um *framework* para computação científica e numérica, nas linguagens de programação Python, Javascript, C/C++ e GO. O foco dele é fornecer recursos pré-construídos e validados para o processamento de dados com redes neurais (aprendizado de máquina), baseado em grafos de fluxo de dados (onde cada nó é um

[Sato 2018].

neurônio e o grafo todo representando a Rede Neural). O *framework* facilita o desenvolvimento de modelos de redes neurais para diferentes tipos de *hardwares*, além de permitir a execução do sistema com ou sem GPUs (*Graphics Processing Unit*) [Souza 2018];

4) ANANCONDA: é uma plataforma que permite gerenciar pacotes, ambientes de desenvolvimento e módulos nas linguagens de programação Python e R, sempre com o foco em na área da Ciência de Dados[4] [Anaconda 2020].

Porém, registra-se que para a RNA Perceptron, tentou-se projetar e implementar os processos de treinamento e testes sem uso de bibliotecas adicionais.

## 3. Metodologia

Este trabalho tem como referência a metodologia Pesquisa por Revisão Bibliográfica apoiada com Estudo de Caso. Já a metodologia de desenvolvimento de sistema é uma versão adaptada de *Scrum* para projetos com dois programadores. Além disso, a técnica *Kanban* é a forma de gerenciar atividades e prazos ao longo deste TFG. Obedecendo as boas práticas de *Scrum*, os encontros (*sprints*) foram semanais sempre com apresentações e discussões das funcionalidades do sistema, com escrita gradativa do texto em paralelo a toda programação do estudo de caso (que envolveu a definição do tipo de RNA, quais atributos e pesos tratar, qual o mecanismo de treinamento).

As tecnologias utilizadas foram a linguagem de programação Python, bibliotecas *numpy*, *wave* (análise de áudio), *mathplot* (gráficos) ambiente de desenvolvimento PyCharm e plataforma ANACONDA.

## 4. Resultados

Dentro do processo de modelagem do sistema, há necessidade de mapear as funcionalidades, já que a metodologia *Scrum* sugere começar por elas. Dessa forma, a Figura 2 ilustra todas as funcionalidades do módulo proposto.

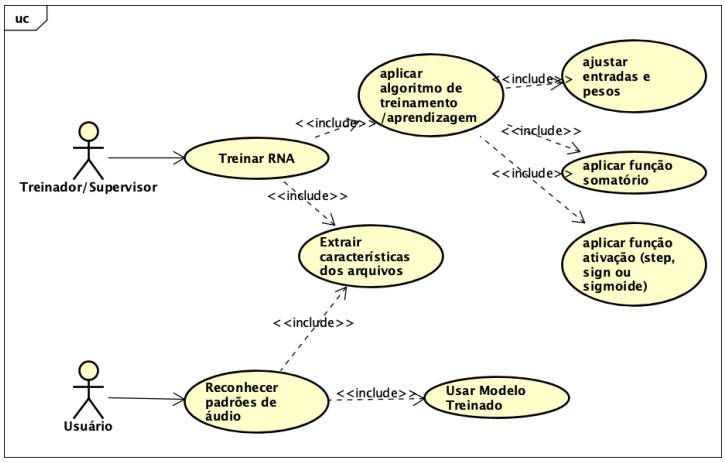



**Figura 2. Diagrama de casos de uso da RNA.**

[4]Campo interdisciplinar que utiliza métodos, processos, algoritmos e sistemas para descoberta de conhecimento em dados estruturados (por exemplo, banco de dados) e não estruturados (por exemplo, páginas Web). Está fortemente relacionada à mineração de dados, aprendizado de máquina e *big data* [Dhar 2013].

### 4.1. Modelagem da RNA classificadora

As características que formam o conjunto de entrada da RNA proposta são: textura de áudio (trechos de áudio); padrão de frequência; padrão de amplitude. Já as saídas esperadas que são utilizadas como treinamento via amostras são: tem ou não violão acústico; tem ou não acordeon.

Neste trabalho, foi utilizado o seguinte formato, um vetor de entradas contendo todas os valores dos atributos; e um vetor de pesos com os pesos respectivos dos atributos. Para o processo de treinamento, também é utilizado um vetor de saída, contendo a presença ou não de determinado instrumento musical. Em relação ao treinamento, pretende-se utilizar treinamento supervisionado com análise de erro e ajustes dos pesos. O erro é calculado desta forma:

$$erro = saida\_esperada - saida\_calculada\_gerada \tag{1}$$

Enquanto o ajuste de peso para cada entrada é:

$$peso = peso + (taxa\_aprendizagem * entrada * erro) \tag{2}$$

A ideia básica do treinamento está de acordo com o ilustrado na Figura 3. Esse algoritmo é o mais usado para calibragem ou ajustes nos pesos sinápticos da RNA.

```
enquanto (saida_gerada != saida_esperada)                  //erro existir
    para cada registro                                     //conjunto de entradas
        calcular a saída com os pesos atuais
        comparar a saida_esperada com a saida_gerada       //calculo erro
        para cada peso da rede
            peso = peso + (taxa_aprendizagem * entrada * erro)  //ajuste pesos

```

**Figura 3. Algoritmo para treinamento supervisionado de RNA.**

No quesito protótipo desta proposta, a RNA é um Perceptron, conforme ilustrado na Figura 4. Há duas listas (*amostras* e *saidas*) que contém as entradas e saídas para o treinamento. A variável *taxa_aprendizagem* indica a probabilidade de aprendizagem (0 a 1, em que 0.1 é 10% de aprendizagem), enquanto que *geracoes* é a quantidade máxima de treinamento que a rede pode executar. A variável *limiar* indica o valor máximo de saída. Finalmente, as variáveis, *n_amostras*, *n_atributos* e a lista de *pesos* para cada peso de cada atributo.

```python
import sys      You, a month ago • estrutura de pastas
import random

class Perceptron:
    ## Primeira função de uma classe (método construtor de objetos)
    ## self é um parâmetro obrigatório que receberá a instância criada
    def __init__(self, amostras, saidas, taxa_aprendizado=0.1, geracoes=1000, limiar=1):
        self.amostras = amostras
        self.saidas = saidas
        self.taxa_aprendizado = taxa_aprendizado
        self.geracoes = geracoes
        self.limiar = limiar
        self.n_amostras = len(amostras) # número de linhas (amostras)
        self.n_atributos = len(amostras[0]) # número de colunas (atributos)
        self.pesos = []
```

**Figura 4. Classe Perceptron.**

O código da Figura 5 mostra o algoritmo clássico de como treinar uma RNA a partir de um conjunto de amostras (contendo as entradas/variáveis/atributos) e as saídas desejadas. Na própria figura do código há comentários explicando cada rotina implementada. Na linha 52 está o código referente à calibragem de cada peso na lista de *pesos* para os *n_atributos* existentes (Figura 5). Na linha 47, há o teste para verificar se a saída gerada combinou com a saída desejada. Caso não ocorra essa combinação, o treinamento precisa ser refeito com a calibragem dos pesos.

```
18      def treinar(self):
19          # Inserir o valor do limiar na posição "0" para cada amostra da lista "amostras"
20          # Ex.: [[0.72, 0.82], ...] vira [[1, 0.72, 0.82], ...]
21          for amostra in self.amostras:
22              amostra.insert(0, self.limiar)
23          # Gerar valores aleatórios entre 0 e 1 (pesos) conforme o número de atributos
24          # a lista de pesos tem tamanho da quantidade de atributos de uma amostra
25          for i in range(self.n_atributos):
26              self.pesos.append(random.random())
27          # Inserir o valor do limiar na posição "0" do vetor de pesos
28          # Ex.: [0.8, 0.1] vira [1, 0.8, 0.1]
29          self.pesos.insert(0, self.limiar)
30          # Inicializar contador de gerações
31          geracoes = 0
32          while True:
33              # Assume-se que o treinamento vai ser eficiente e numa geração o algoritmo possa aprender
34              aprendeu = True
35              # Para cada amostra
36              for i in range(self.n_amostras):
37                  # Inicializar potencial de ativação
38                  soma = 0
39                  # Para cada atributo
40                  for j in range(self.n_atributos + 1):
41                      # Multiplicar amostra e seu peso e também somar com o potencial que já tinha
42                      soma += self.pesos[j] * self.amostras[i][j]
43                  # Obter a saída da rede considerando a função sinal
44                  saida_gerada = self.funcao_ativacao_signal(soma)
45                  # Verificar se a saída da rede é diferente da saída desejada
46                  # se sim, calibrar ou treinar ou ajustar ou fazer aprender
47                  if saida_gerada != self.saidas[i]:
48                      # Calcular o erro
49                      erro = self.saidas[i] - saida_gerada
50                      # Fazer o ajuste dos pesos para cada elemento da amostra
51                      for j in range(self.n_atributos + 1):
52                          self.pesos[j] = self.pesos[j] + self.taxa_aprendizado * erro * self.amostras[i][j]
53
54                      # se entrou no if é porque ainda não aprendeu
55                      aprendeu = False
56              geracoes += 1
57              if aprendeu or geracoes > self.geracoes:
58                  print('Quantidade de gerações para aprender: %d\n' % geracoes)
59                  break
```

**Figura 5. Método *treinar* da classe Perceptron.**

## 4.2. Tratamento dos arquivos de áudio

Na Figura 6, a linha 7 abre o arquivo de áudio .wav. A linha 11 retorna o número de canais armazenados no arquivo de áudio. A linha 15 retorna o número de bytes por amostra. A linha 19 retorna a taxa de amostragem com que foram gravados os dados de áudio. A linha 22 retorna a quantidade de dados armazenados. A linha 25 retorna o tipo de compactação utilizada. A linha 29 retorna uma *string* binária com todos os bytes armazenados na área de dados do arquivo .wav. A linha 22 define o intervalo de amostragem em segundos. Por fim, a linha 35 cria um *array numpy* que conterá os pontos de amostragem distribuídos no tempo.

Na Figura 7, é realizado a conversão da *string* binária lida através de "*readframes*" para uma lista Python, onde na linha 39 é criado esta lista, na linha 40 é exibido o tamanho da variável *frames*, na linha 42 tem um laço de repetição que percorre o vetor de *frames* a cada duas posições e, na linha 43 os *frames* são convertidos para *bytes* e adicionados a lista. A linha 48 converte a lista para um *array numpy*. A linha 58 prepara para traçar o gráfico dos primeiros 100 pontos do *array*. A linha 55 insere a grade na tela. A linha 58 mostra o gráfico. Por fim, a linha 61 fecha o arquivo de áudio.

Como resultado da execução desse algoritmo, tem-se o gráfico da Figura 8, que exibe os 100 primeiros pontos do *array* gerado na Figura 7, onde o eixo x representa o

src2 > abrindo_arquivo_de_audio.py

lucaslima2018, 2 minutes ago | 1 author (lucaslima2018)

```python
import matplotlib.pyplot as plt      lucaslima2018, 19 hours ago • Abrir Arquivos de Áudio em Python
import numpy as np
import wave
import struct
def main(args):
    '''Abrir o arquivo “exemplo1.wav”, com o comando open do pacote wave no modo leitura'''
    arquivoWave = wave.open('exemplo1.wav', 'r')

    '''A função “getnchannels()” retorna o número de canais armazenados no arquivo wav. No exemplo
    somente arquivos mono serão processados'''
    print("Número canais: ", arquivoWave.getnchannels())

    '''A função “getsamplewidth()” retorna o numero de bytes por amostra. No exemplo apenas 2 bytes por
    amostra'''
    print("Número bytes: ", arquivoWave.getsampwidth())

    '''A função “getframerate()” retorna a taxa de amostragem (amostras por segundo) com que foram
    gravados os dados wav.'''
    print("Taxa de amostragem: ", arquivoWave.getframerate())

    '''A função “getnframes” retorna o número de frames, ou seja, a quantidade de dados armazenados.'''
    print("Número de frames: ", arquivoWave.getnframes())

    '''A função “getcompname()” retorna o tipo de compactação. Nenhuma compactação foi usada.'''
    print("Compactação: ", arquivoWave.getcompname())

    '''A função “readFrames(nFrames)” retorna uma string binária com todos os bytes armazenados na
    área de dados do arquivo wav.'''
    frames = arquivoWave.readframes(arquivoWave.getnframes())

    '''DeltaX é o intervalo de amostragem, em segundos.'''
    deltaX = 1.0 / arquivoWave.getframerate()

    '''Criar array numpy que conterá os pontos de amostragem distribuídos no tempo.'''
    tempo = np.arange(start=0, stop=arquivoWave.getnframes() * deltaX, step=deltaX, dtype=np.float)
```

**Figura 6. Parte 1 do Código Python que abre arquivo e captura algumas propriedades de áudio.**

```python
    '''Converter string binária lida através de “readframes” para uma lista python.'''
    waveDataList = []
    print('Tamanho da variavel frames: ',len(frames))

    for nLoop in range(0, len(frames), 2):
        waveDataList.append(struct.unpack('<h', (frames[nLoop] + frames[nLoop + 1]).to_bytes(2,'big')))

    #waveDataList = [struct.unpack("<h", frames[nLoop] + frames[nLoop + 1])[0] for nLoop in range(0, len(frames), 2)]

    '''Converter a lista python para um array numpy, mais adequado para processamento matemático.'''
    waveArray = np.array(waveDataList)

    '''Preparar para traçar o gráfico dos primeiros 100 pontos do array. 100 pontos foram selecionados
    para apresentar cerca de dois ciclos completos na tela.'''
    plt.plot(tempo[0:100], waveArray[0:100])

    '''Inserir grade na tela'''
    plt.grid()

    '''Mostrar o gráfico. Para fechar pressione o “x” no canto superior direito.'''
    plt.show()

    '''Fechar arquivo wav.'''
    arquivoWave.close()
    return 0

# if __name__ == '__main__':
#     import sys
#
#     sys.exit(main(sys.argv))

main()
```

**Figura 7. Parte 2 do Código Python que trata as propriedades capturadas na Parte 1.**

tempo e o eixo y a taxa de amostragem formada pelos *frames* extraídos do arquivo de áudio.

### 4.3. Amostras de áudio para treinamento RNA

Foram realizadas gravações de 6 áudios contendo instrumentos musicais específicos em cada áudio. Todos os áudios têm formato *.wav*, com um tempo médio de 8 segundos e com os instrumentos Acordeon e Violão. Assim sendo, para a captura do áudio foram usados:

- Microfone interno do Notebook Acer, processador Core i5, memória RAM de 8 GB, placa integrada de áudio e sistema operacional Windows 10 Home;
- Software Audacity 2.1.3;
- Arquivos no formato *.wav*;

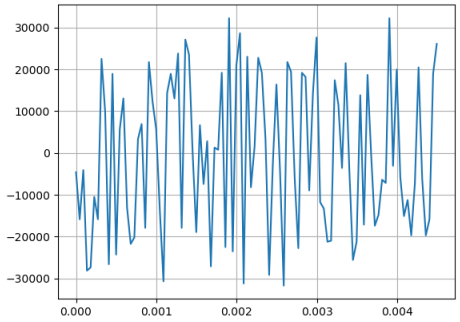



**Figura 8. Gráfico dos 100 primeiros pontos de amostra do arquivo de áudio exemplo.**

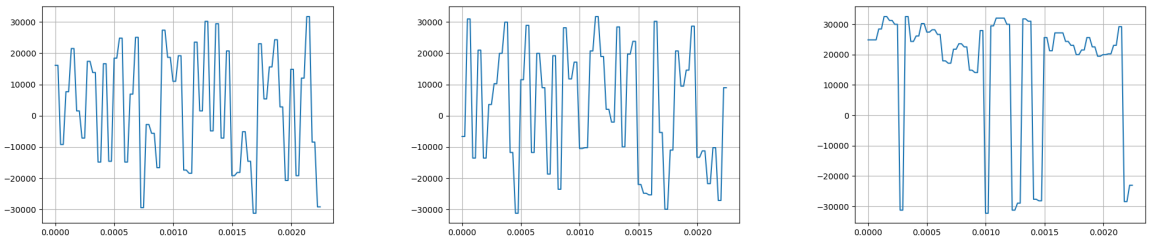
- Violão de aço da marca Tagima Kansas em modo acústico[5];
- Acordeon pianado 120 baixos da marca Michael, também em modo acústico.

Registra-se que o violão acústico está afinado em Mi maior (afinação tradicional), enquanto o acordeon está na sua afinação tradicional de fábrica. Os arquivos produzidos (2 repetições de 4 tempos), como amostras para o treinamento da RNA, foram os seguintes:

- arquivo 1 - violão com acordes de Dó maior, Lá menor, Si sustenido menor, Mi menor;
- arquivo 2 - violão com acordes de Sol maior, Ré com baixo em Fá sustenido, Fá maior, Dó maior;
- arquivo 3 - violão com acordes de Lá, Fá sustenido menor, Mi maior, Ré maior;
- arquivo 4 - acordeon com acordes de Ré, Lá, Sol, Lá;
- arquivo 5 - acordeon com acordes de Sol, Ré, Dó, Ré;
- arquivo 6 - acordeon com acordes de Fá sustenido maior, Dó sustenido em sétima, Si, Dó sustenido em sétima.

Na Tabela 1, há os gráficos dos arquivos 1, 2 e 3 referentes a áudios com violão acústico.

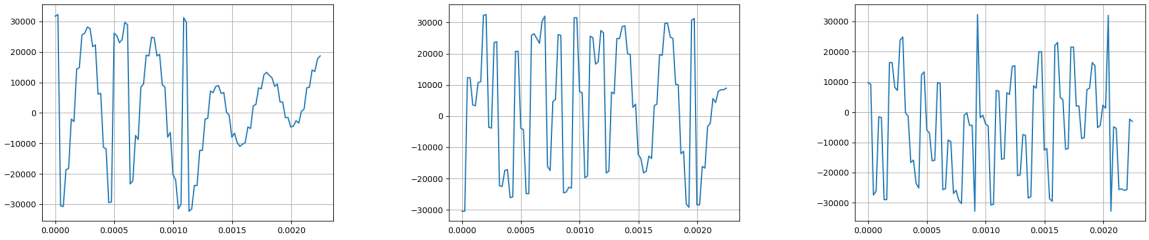
**Tabela 1. Gráficos dos 3 primeiros arquivos com violões capturados.**




Já na Tabela 2, há os gráficos dos arquivos 4, 5 e 6, que são referentes aos áudios com acordeon.

[5]Sem captação eletrônica.

**Tabela 2. Gráficos dos últimos 3 arquivos de acordeons capturados.**

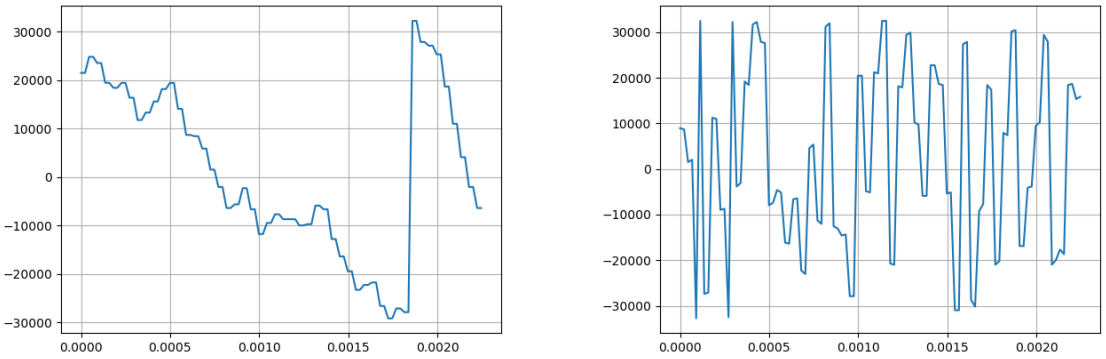


Registra-se que os resultados de cada instrumento possuem um padrão comum, que é possível visualizar nas imagens (amplitude e frequência). Contudo, na Tabela 4, há todas as propriedades extraídas de cada arquivo de áudio.

## 4.4. Arquivos de áudio para reconhecimento via RNA

Nesta seção, há dois arquivos .wav com a música Canto Alegretense, uma em violão, outra no acordeon. Os gráficos das propriedades estão na Tabela 3.

**Tabela 3. Gráficos dos dois arquivos de áudio contendo Canto Alegretense (somente instrumental). Primeiro é referente ao violão, enquanto o segundo, ao acordeon.**



Finalmente, há todos os atributos mapeados, tanto dos áudios de treinamento, quanto dos áudios para reconhecimento, que estão ilustrados na Tabela 4.

**Tabela 4. Tabela contendo as propriedades dos arquivos de áudio.**

| Propriedades / Amostras | Violão | | | | Acordeon | | | |
|---|---|---|---|---|---|---|---|---|
| | Arquivo 1 | Arquivo 2 | Arquivo 3 | Música Exemplo | Arquivo 4 | Arquivo 5 | Arquivo 6 | Música Exemplo |
| Número de canais | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| Número de bytes | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| Taxa de amostragem | 44100 | 44100 | 44100 | 44100 | 44100 | 44100 | 44100 | 44100 |
| Número de frames | 909945 | 977139 | 676921 | 1410141 | 318976 | 296960 | 304640 | 2490368 |
| Compactação | sem compactação | sem compactação | sem compactação | sem compactação | sem compactação | sem compactação | sem compactação | sem compactação |
| Tamanho variável frames | 3639780 | 3908556 | 2707684 | 5640564 | 1275904 | 1187840 | 1218560 | 9961472 |

## 4.5. Códigos para extração de atributos de entrada

Como discutido, uma RNA precisa de uma lista de entrada contendo os atributos necessários e uma lista de saída (com o que é esperado da entrada), tanto para o treinamento quanto para o processo de reconhecimento.

Dessa forma, a classe Entrada (Figura 9) foi criada para guardar esses atributos para a lista de entrada.

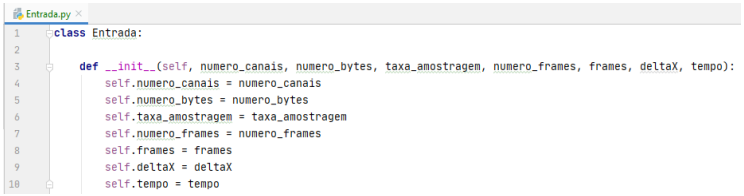



**Figura 9. Classe Entrada com os atributos de entrada para a RNA.**

A Figura 10 mostra o método em que os arquivos de áudio são tratados. A linha 4 mostra a assinatura do método. Na linha 5, é criado uma lista chamada 'lista-arquivos-amostras', onde nas linhas 6, 7, 8 são adicionadas a essa lista as amostras de áudio contendo violão e, nas linhas 9, 10 e 11 são adicionados as amostras de áudio contendo acordeon. Na linha 12 é criado uma segunda lista denominada 'lista-entradas'. Na linha 13 é criada a 'lista-saidas', contendo as saídas das respectivas amostras. Na linha 15, há uma repetição que irá percorrer a 'lista-arquivos-amostras' e, para cada amostra de áudio irá exibir os atributos definidos nas linhas 16 à 22. Na linha 23, serão calculados os valores de amplitude e frequência. Na linha 24, a variável 'dados-entrada' irá armazenar os atributos anteriormente informados, definido na classe 'Entrada'. Na linha 25, a variável 'dados-entrada' será guardada na 'lista-entradas'.


**Figura 10. Método que faz o tratamento de arquivos de áudio.**

Registra-se que o atributo frequência também é conhecido como frequência de onda. E é uma medida do número total de vibrações ou oscilações feitas dentro de um período determinado de tempo. Já amplitude de onda é a distância que vai do seu eixo até o seu ponto mais elevado (crista da onda). O comprimento de onda equivale à distância entre duas cristas consecutivas, dois vales consecutivos ou dois pontos correspondentes consecutivos da onda.

## 4.6. Análise e discussão dos resultados

Depois das implementações dos dois processos: treinamento da RNA (via 6 amostras de arquivos .wav ) e reconhecimento de áudios, percebe-se que a qualidade das amostras é fundamental, pois são elas que habilitam a RNA para o reconhecimento.

Outro ponto importante, é definir objetivamente os atributos que devem ser usados como atributos de entrada da RNA, seja para treinamento, seja para reconhecimento. E é neste ponto que o trabalho apresenta sua maior fragilidade, uma vez que os valores de frequência e amplitude ainda são obscuros no processo projetado e implementado. Dessa forma, a qualidade do processo de treinamento fica debilitado, logo o processo de reconhecimento da RNA também o fica.

Destaca-se que as entradas foram padrão frequência e de amplitude, como é possível visualizar nos gráficos das figuras. Na Figura 11, destacou-se alguns dados de todos os gráficos gerados, como picos, vales e extensão da onda. Mas nenhum desses podem ser utilizados para treinar e testar a RNA.

| | picos | intervalo | max | min | acima 30000 | abaixo -30000 |
|---|---|---|---|---|---|---|
| aquivo1 | 52 | 60000 | 30000 | -30000 | 1 | 1 |
| arquivo2 | 52 | 60000 | 30000 | -30000 | 2 | 1 |
| arquivo3 | 54 | 60000 | 30000 | -30000 | 7 | 3 |
| | | | | | | |
| arquivo4 | 55 | 60000 | 30000 | -30000 | 2 | 3 |
| arquivo5 | 55 | 60000 | 30000 | -30000 | 4 | 1 |
| arquivo6 | 52 | 60000 | 30000 | -30000 | 2 | 3 |
| | | | | | | |
| cantoalegretense violao | 49 | 60000 | 30000 | -30000 | 1 | 0 |
| cantoalegretense acordeon | 52 | 60000 | 30000 | -30000 | 6 | 3 |

**Figura 11. Dados dos gráficos extraídos manualmente.**

## 5. Conclusões

Este trabalho discutiu a análise de padrões em arquivos digitais de áudio, por meio Redes Neurais Artificiais. Nesta etapa, foram apresentados conceitos fundamentais relacionados a esses temas. Para a compreensão da área e de técnicas utilizadas em extração de características para posterior reconhecimento, pesquisou-se trabalhos relacionados, que apontaram métodos e ferramentas interessantes para seguir neste trabalho. Dessa forma, foi possível mapear o mecanismo de treinamento do classificador em Python.

Também foi possível definir a RNA, ou seja um Perceptron com treinamento supervisionado. A quantidade de entradas (atributos ou variáveis) pode ser qualquer número, visto que o Python dá condições do trabalho de listas dentro de listas. Assim, uma amostra com *n_atributos* vai ser inserida na lista *amostra* e o código pode identificar essa quantidade (como ilustrado do comando *for* das linhas 36 e 51 da Figura 5).

O projeto e a implementação de RNAs tem sempre um desafio, definir com precisão quais serão os atributos de entrada da RNA e se será preciso mais de um neurônio (mais camadas). Em tese, cada neurônio recebe entradas e seus pesos que pertençam a uma mesma categoria. Dessa forma, pode ser necessário vários neurônios para os diferentes tipos (categorias) de entradas. Para o processamento de arquivos na RNA projetada, os áudios não poderiam conter nenhum tipo de compressão de dados. Além disso, a manipulação de arquivos de áudio não é simples, ainda mais quando se trata em definir atributos (propriedades) para os vetores de entrada de uma RNA e, logo, seus respectivos pesos. Portanto, extrair a frequência e a amplitude para que fossem utilizados na RNA é necessário outros recursos computacionais. Como neste trabalho o vetor de entrada e o vetor de pesos são muito grandes, ou seja, muitas posições, o uso de computação tradicional (via instrução de controle $for$), pode aumentar consideravelmente o processamento. Portanto, refatorar este módulo com Tensorflow, que faz uso de recursos de paralelismo

e computação concorrente, poderia agilizar consideravelmente os processos. Então, uma comparação destes algoritmos implementados totalmente em Python (sem bibliotecas ou pacotes adicionais) com uma versão implementada e analisada via Tensorflow seria um bom trabalho futuro.

## Referências


Almeida, U. N. (2009). Reconhecimento de ritmo musical por anÁlise de sinais de Áudio de amplitude e frequÊncia. Monografia, Ciência da Computação no Unilassale - Canoas, RS.

Anaconda (2020). *Data science technology for groundbreaking research. a competitive edge. a better world. human sensemaking*. https://www.anaconda.com, Acessado em Maio de 2020.

Benite, M. (2003). *Aplicação de modelos de redes neurais na elaboração e análise de cenários macroeconômicos*. PhD thesis, Universidade de São Paulo.

Borges, L. E. (2014). *Python para Desenvolvedores: Aborda Python 3.3*. Novatec Editora.

Canaltech (2020). *O que é WAV ou WAVE?* https://canaltech.com.br/produtos/O-que-e-WAV-ou-WAVE/, Acessado em Dezembro de 2020.

Carneiro, L. (2012). Desenvolvimento de um sistema de reconhecimento automático de músicas-SRAM. Monografia, Universidade de Caxias do Sul - UCS.

Comunidade de Desenvolvedores (2020). Numpy. Disponível em https://numpy.org. Acessado em maio de 2020.

Costa, Y. M. et al. (2013). Reconhecimento de gêneros musicais utilizando espectrogramas com combinaçao de classificadores.

Dhar, V. (2013). Data science and prediction. *Communications of the ACM*, 56(12):64–73.

Lima, F. G., Perera, L. C. J., Kimura, H., and da Silva Filho, A. C. (2009). Aplicação de redes neurais na análise e na concessão de crédito ao consumidor. *Revista de Administração-RAUSP*, 44(1):34–45.

Logan, B. et al. (2000). Mel frequency cepstral coefficients for music modeling. In *Ismir*, volume 270, pages 1–11.

Muniz, D. N. (2009). Estudo sobre Reconhecimento de Áudio Repetitivo: desenvolvimento de um protótipo. Monografia, Sistemas de Telecomunicações do Instituto Federal de Santa Catarina.

Nogueira, A., Azevedo, J., Baptista, V., and Siqueira, S. (2006). Um *overview* sobre Reconhecimento de Padrões. Artigo, Associação Educacional Dom Bosco.

Russel, S., Norvig, P., et al. (2013). *Artificial intelligence: a modern approach*. Pearson Education Limited.

Santos, M. A., Souza, D. H. S. d., Penedo, A. S. T., and Silveira Martins, E. (2018). Aplicação de redes neurais no brasil: um estudo bibliométrico:: Brapci 2.0. *Biblionline; v. 12, n. 2 (2016); 101-116*, 24(2):116–101.

Sato, L. C. (2018). Deep learning na segurança computacional: detecção inteligente de códigos maliciosos. Monografia de graduação, Universidade Tecnológica Federal do Paraná.

Shinohara, V. Y. (2018). Classificação automática de música utilizando aprendizagem de padrões de votação. Tese, Universidade Tecnológica Federal do Paraná.

Souza, E. F. (2018). *Tensorflow*. https://medium.com/trainingcenter/tensorflow-cbe1595a49e3, Acessado em Maio de 2020.