UNIVERSIDADE POSITIVO
ESCOLA POLITÉCNICA
ENGENHARIA ELÉTRICA

MARCOS DANIEL SANTANA
MARCUS VINÍCIUS ALBANI

JARBAS: UM ASSISTENTE VIRTUAL POR COMANDO DE VOZ PARA AUTOMAÇÃO RESIDENCIAL

TRABALHO DE CONCLUSÃO DE CURSO

CURITIBA
2018

MARCOS DANIEL SANTANA
MARCUS VINÍCIUS ALBANI

JARBAS: UM ASSISTENTE VIRTUAL POR COMANDO DE VOZ PARA AUTOMAÇÃO RESIDENCIAL

Trabalho de Conclusão de Curso apresentado ao Curso de Engenharia Elétrica da Universidade Positivo como requisito parcial para obtenção do título de Bacharel em Engenharia Elétrica.

Orientador: Prof. Leonardo G. Tavares, MSc.

CURITIBA
2018

RESUMO

SANTANA MARCOS D.; ALBANI, M. V. *JARBAS: um assistente virtual por comando de voz para Automação Residencial*. Trabalho de conclusão de curso, 2018.

A tecnologia de automação e a inteligência artificial (IA) têm avançado muito nos últimos anos, não apenas no ramo industrial, mas também em ambientes residenciais. Nesses últimos, as aplicações são diversas e como, por exemplo, pode-se citar a identificação da presença de pessoas não autorizadas, o monitoramento do consumo de energia, o acionamento de eletrodomésticos via comando de voz e muitas outras. Neste trabalho foi desenvolvido um assistente que consiste de um tradutor homem/máquina para comandos curtos de voz que, auxiliado por uma Rede Neural Artificial do tipo Multilayer Perceptron, interpreta comandos de voz e envia a tradução destes para eletrodomésticos e equipamentos específicos através da rede Wi-Fi da residência. O assistente é ativado através do comando "Jarbas", que é o nome que lhe foi atribuído. Depois de ativar o assistente, o usuário pode comandar os eletrodomésticos da casa através de comandos curtos como, por exemplo, "ligue tv" ou "desligue ar-condicionado". Nesta primeira versão do assistente foram programados dez comandos ("Jarbas", "Ligue", "Desligue", "Música", "Sala", "Quarto", "Cozinha", "Ar-condicionado", "Tv" e "Café") a fim de viabilizar a execução dos testes e a prova do conceito. Para a construção desta base de dados de comandos, foram gravadas locuções dos dez comandos escolhidos por um grupo de 62 voluntários, incluindo homens, mulheres e crianças. Além dos exemplos gravados, foram utilizadas técnicas de *Data Augmentation* para ampliar a base de dados com o intuito de aumentar a robustez da Rede Neural treinada. Testes realizados com o equipamento comprovam a sua aplicabilidade dentro do contexto proposto. A expansão do vocabulário de comandos pode ser feita repetindo a mesma metodologia utilizada nesta primeira versão.

Palavras-chaves: Automação Residencial, Rede Neurais Artificiais, Comandos curtos de voz.

ABSTRACT

Automation technology and artificial intelligence (AI) have advanced a lot in recent years, not only in the industrial branch, but also in residential environments. In these ones, the applications are diverse, for example, the identification of unauthorized persons, the monitoring of energy consumption, the activation of appliances by voice command and many others. In this work, an assistant was developed consisting of a man/machine translator for short voice commands that, aided by a Artificial Neural Network Multilayer Perceptron, interprets the voice commands and sends these translation to home appliances equipment by residences Wi-Fi network. The assistant is activated through the "Jarbas" command, which is the name that of the assistant. After activating the assistant, the user can control household appliances through short commands such as "turn on TV" or "turn off air conditioning". In this first version of the assistant, ten commands was programed (“Jarbas”, “Ligue”, “Desligue”, “Música”, “Sala”, “Quarto”, “Cozinha”, “Ar-condicionado”, “Tv” e “Café”) in order to make feasible the execution of tests, and prove the concept. For building this database of commands, were recorded voice of these ten commands by a group of 62 volunteers, including men, women and children. In addition to the recorded examples, data Augmentation techniques were used to extend the database, in order to increase the robustness of the trained Neural Network. Tests performed with the equipment prove their applicability within the proposed context. The expansion of the vocabulary of commands can be done repeating the same methodology used in this first version.

Key-words:Residential Automation, Artificial Neural Network, Short Voice Commands.

LISTA DE ILUSTRAÇÕES

SUMÁRIO

1 INTRODUÇÃO

Em latim, automação significa mover-se por si, o qual é um sistema baseado em computador, utilizado amplamente nas indústrias, reduzindo os custos, aumentando a qualidade e a segurança dos processos e dos produtos (MORAES; CASTRUCCI, 2007).

No ramo domiciliar, essa automação se chama "automação residencial"e é responsável por facilitar o dia a dia das pessoas. A automação residencial pode executar tarefas mais básicas, como por exemplo, ligar uma lâmpada em um horário estabelecido, até tarefas mais específicas, como monitorar a presença de indivíduos não autorizados.

A voz é a principal forma de comunicação entre a humanidade desde os primórdios da civilização. Por meio da fala, o ser humano pôde comunicar-se e evoluir até os dias atuais, onde a tecnologia tem se mostrado muito útil nesta área.

No meio dessa evolução, a voz tem sido usada como um importante recurso de biometria e como comandos para execução de tarefas. O reconhecimento de comandos de voz vem mostrando-se cada vez mais eficiente e mais utilizado, principalmente em *smartphones*. Assim, o uso de comandos utilizando a voz na automação residencial pode trazer muitos benefícios, comandando equipamentos com um simples comando curto de voz.

1.1 PROBLEMA

Pessoas com limitação motora, muitas vezes precisam de ajuda de terceiros para realização de tarefas básicas, como por exemplo, acender uma lâmpada, abrir a janela, ligar um aparelho elétrico, regular a temperatura de um chuveiro, entre outros.

A automação inclusiva busca soluções que garantam a acessibilidade, segurança, conforto e saúde de seus usuários proporcionando o bem-estar, em uma sociedade que ainda não possui normas abrangentes para a automação residencial, tendo que utilizar normas estrangeiras, como a do Instituto Americano de Normas e Padrões, a NBR14565 (GUEDES *et al.*, 2012)

Com a automação tornando-se cada vez mais comum, a execução de tarefas está ficando cada vez mais fácil e rápida, assim, atividades que muitas vezes não eram realizadas por falta de tempo, ou simplesmente por indisponibilidade passam a ser foco para o desenvolvimento. Os estudos nessa área possibilitam um avanço na tecnologia, tanto residencial quanto industrial, para ROCKENBACH (2004),

> proporcionando uma maior flexibilidade, eficiência, rapidez onde se tem uma otimização da produção. Com toda esta evolução temos o lado oposto onde o trabalho humano se torna obsoleto em algumas funções, tendo uma diminuição do mercado de trabalho. (2004,p.29).

1.2 JUSTIFICATIVA

Nos últimos trinta anos, a tecnologia vem se desenvolvendo num fluxo cada vez mais acelerado, muitos dispositivos de automação inicialmente foram considerados como um luxo devido ao alto valor de mercado para implantar-se em residências. Após anos de avanços dos estudos na área de tecnologia como um todo, foi observado que é possível automatizar residências para ajudar pessoas idosas e com deficiências.

Pode-se notar que a população idosa brasileira tem crescido nos últimos dez anos, podendo a chegar a 38 milhões em 2027 [1] , tendo uma taxa crescente do envelhecimento segundo a Figura 1. Assim como dados do IBGE[2] mostram que o Brasil tem 6,2% de pessoas com algum tipo de deficiência, desta maneira foi pensado no desenvolvimento de um dispositivo por comando de voz, para propiciar um auxilio no dia-a-dia de pessoas com idade avançada e pessoas com necessidades motoras.

Um sistema que controle uma carga através de um comando de voz apresenta inúmeras vantagens, como por exemplo, a ativação de um equipamento apenas com um comando de voz, do tipo; “Tv Ligue”, ou “Ar Condicionado Ligue”, não tendo a necessidade de apertar um ou mais botões de um controle remoto. Outro ponto positivo, é a praticidade de não precisar se deslocar do local em que você se encontra no momento, facilitando também para pessoas que não conseguem locomover-se com facilidade.

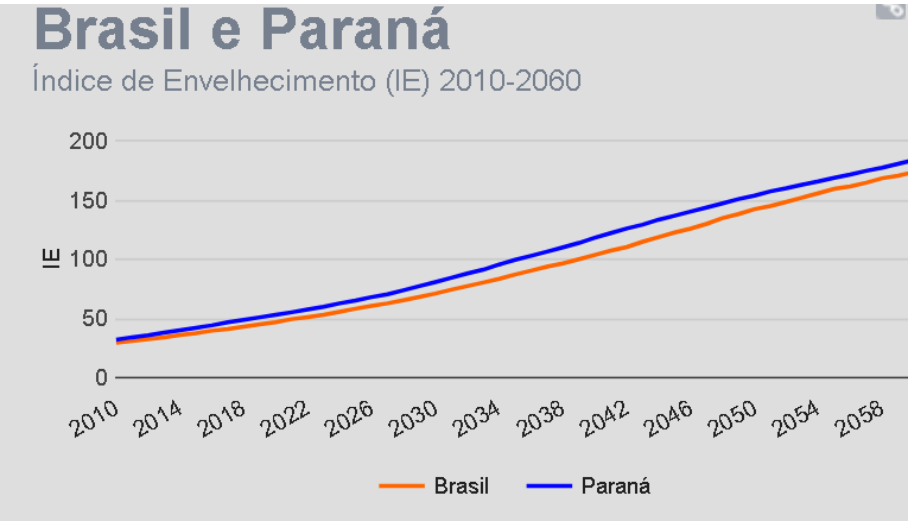


Figura 1 – Índice de Envelhecimento (IE): Número de idosos para cada 100 pessoas menores de 15 anos de idade, no ano considerado:
Fonte:Instituto Brasileiro de Geografia e Estatística(IBGE)

[1] Disponível em: <https://sebraeinteligenciasetorial.com.br/produtos/boletins-de-tendencia/hoteis-residenciais-para-a-terceira-idade-novo-modelo-de-negocio/5a8f0fbf5e3cff1a007c8710 >. Acesso em 03 abr.2018.

[2] Disponível em: <http://agenciabrasil.ebc.com.br/geral/noticia/2015-08/ibge-62-da-populacao-tem-algum-tipo-de-deficiencia>. Acesso em 03 abr.2018

1.3 OBJETIVOS

Nesta seção serão abordados os objetivos gerais e específicos do trabalho, relativo ao problema anteriormente apresentado.

1.3.1 Objetivo Geral

O objetivo central deste trabalho é o desenvolvimento de um protótipo de um sistema embarcado, capaz de interpretar comandos curtos de voz, e enviar uma tradução destes comandos através de uma rede WiFi, utilizando o protocolo de comunicação MQTT.

1.3.2 Objetivo Específico

Os objetivos específicos deste trabalho são:

- Desenvolver os circuitos e o software do dispositivo de comando de voz;
- Elaborar um algoritmo classificador baseado em aprendizagem de máquina;
- Criar um banco de dados com os comandos de voz;
- Avaliar e testar conjuntos de atributos descritores do sinal de voz;
- Enviar os comandos via WiFi utilizando o protocolo MQTT.

2 FUNDAMENTAÇÃO TEÓRICA

Nesta seção será apresentada uma revisão teórica dos conceitos usados para o desenvolvimento do trabalho, tais como, as Redes Neurais Artificiais e os extratores de características.

2.1 INTELIGÊNCIA COMPUTACIONAL

Uma das características da inteligência é a capacidade de distinguir e generalizar padrões, ficando mais aguçada cada vez que se aprende novos padrões (PIAZZI, 2015). Por exemplo, um ser vivo nasce com uma inteligência muito pobre, e vai enriquecendo sua inteligência na medida que vai aprendendo novos padrões.

Caso seja solicitado à uma criança para pegar uma bola, e essa criança não conheça uma bola, ela não saberá o que fazer, pois ela não conhece aquele padrão. Mas basta alguém mostrar uma bola que ela logo distingue o padrão arredondado à palavra 'bola'. Com o padrão arredondado associado à palavra bola, a criança ainda pode confundir uma bola com uma melancia por exemplo. Então será necessário ela aprender o padrão; arredondado, pesado, com listras verde claras e verde escuras, associado à palavra melancia. Dessa forma a inteligência dessa criança ficará mais precisa para analisar objetos arredondados por exemplo.

A inteligência artificial possibilita a capacidade de uma máquina aprender um padrão. Esses padrões podem ser aprendidos através de alguns conceitos que funcionam baseados no cérebro, como por exemplo; árvore de decisões, onde os dados fornecidos à entrada dessa árvore passam por alguns blocos, que analisam se aquela informação é falsa ou verdadeira de acordo com um referencial, mandando ou não para os blocos seguintes, simulando nossos neurônios.

Outra forma de uma IA aprender um padrão, é usando o método de modelo oculto de Markov (Hodden Markov Model - HMM). De acordo com CARVALHO (2013), os HMMs podem ser vistos como máquinas de estados finitos, onde a cada unidade de tempo ocorre uma transição de estado e, a cada estado se emite um vetor com uma função densidade de probabilidade associada, parecido com nossas sinapses.

Há outras técnicas para analisar padrões, como por exemplo, o método do vizinho mais próximo (K Nearest Neighbor, KNN), redes neurais, entre outros. Para o desenvolvimento desse trabalho, será usado o conceito de rede neural artificial (RNA), que reúne características usadas na árvore de decisões e de HMM para generalizar um padrão, possuindo uma estrutura maciçamente paralelamente distribuída e habilidade de aprender, e, portanto, generalizar padrões (HAYKIN, 2001). Essa generalização se refere ao fato da RNA apresentar saídas adequadas para o padrão analisado.

2.2 APRENDIZADO SUPERVISIONADO E NÃO SUPERVISIONADO

Toda rede neural passa por um processo de aprendizagem similar ao cérebro humano, como descrito no tópico anterior. Na aprendizagem de uma RNA, há parâmetros livres, denominados de pesos sinápticos e bias, que são adaptados através de um processo de estimulação, proporcionado pelo ambiente no qual a rede neural está inserida. A maneira que ocorre a modificação dos parâmetros livres define o tipo de aprendizagem, que para Bishop (2006), pode ser não supervisionada ou supervisionada para tornar a rede mais instruída sobre seu ambiente, após cada interação do processo de aprendizagem.

Para a RNA aprender o conhecimento acerca do ambiente, ou seja, aprender um padrão, é utilizado um conjunto de exemplos rotulados, onde o sinal de entrada de um exemplo é associado à uma resposta desejada na saída, ou com exemplos não rotulados, onde não há relação alguma com a saída, penas com a entrada (HAYKIN, 2001).

Informações prévias e invariâncias às transformações do sinal de entrada devem ser adicionadas ao projeto de RNA, afim de não ter que aprendê-las, simplificando com isso o projeto da RNA.

2.2.1 Aprendizado Supervisionado

No processo de aprendizagem supervisionado é utilizado um agente externo (Professor), que aponta à rede a resposta desejada em função de dados de entrada, através de exemplos rotulados. Neste caso o professor supervisiona e indica explicitamente o comportamento bom ou ruim, onde se utiliza da diferença entre a resposta desejada e a saída real da rede para determinar o comportamento da aprendizagem durante o treinamento (SANT'ANA, 2018).

A diferença entre a resposta esperada e a resposta real obtida pela rede é chamada de sinal de erro. A ação combinatória do sinal de erro com o sinal de entrada, ajusta os parâmetros livres a cada ciclo de treinamento, chamado de época, como ilustrado na figura 2, desta forma o conhecimento do ambiente, disponível ao professor, é transferido para a rede da forma mais completa possível, através do treinamento (HAYKIN, 2001). Este tipo de algoritmo de treinamento é conhecido como método de aprendizagem por correção de erro.

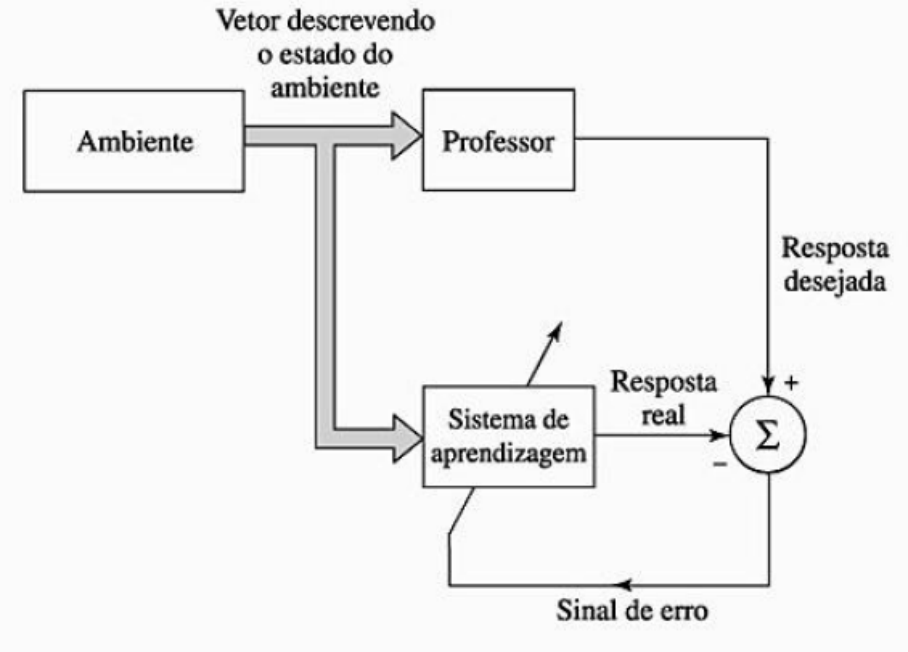


Figura 2 – Diagrama em bloco do processo de aprendizagem supervisionada, com método de correção de erro.

Fonte: (HAYKIN, 2001).

O processo de aprendizagem é finalizado quando a variação dos pesos sinápticos ao longo de cada ciclo é mínima, desta forma se pode dispensar o uso do professor e deixar a rede neural interpretar o ambiente por conta própria.

2.2.2 Aprendizado não-supervisionado

Conhecida também como aprendizagem auto organizada. Nesse processo de aprendizagem, a rede tem que definir atributos estatísticos relevantes para alterar os parâmetros livres, tendo que desenvolver uma representação própria dos estímulos que entram na RNA, fazendo uso apenas de exemplos não rotulados, segundo RAUBER (2004).

Neste método de aprendizagem não existe um "Professor" para supervisionar a aprendizagem, apenas padrões de entrada estão disponíveis para RNA. A Rede Neural processa as entradas, detectando suas regularidades, e tenta progressivamente estabelecer representações internas para codificar características e classificá-las automaticamente, existindo redundância nos dados de entrada, para que se consiga encontrar padrões em tais dados (PRADO, 2011).

Este tipo de aprendizagem faz uso da regra da aprendizagem competitiva, onde apenas um neurônio, denominado neurônio vencedor, é ativado em sua camada. A ativação do neurônio vencedor é definida em relação ao potencial de ativação[1] , o neurônio com maior valor do potencial de ativação daquela camada é ativado. Uma realimentação entre todos os neurônios de

[1] O potencial de ativação é a saída de um neurônio artificial, leia o tópico 2.3.1 Neurônio artificial, para mais informações.

uma mesma camada, faz a saída dos demais neurônios (neurônios perdedores) daquela mesma camada serem desativados. A camada onde essa competição acontece é chamada de camada de competição. Nesse processo de aprendizagem nem todas as camadas precisam ser competitivas.

Os métodos de aprendizagem baseado em memória, aprendizagem Hebbiana e aprendizagem competitiva, são exemplos de aprendizagem não supervisionada.

2.3 REDE NEURAL ARTIFICIAL (RNA)

A RNA foi projetada para modelar a maneira como o cérebro realiza uma determina função, sendo ela simulada por programação ou utilizando componentes eletrônicos, obtendo uma grande habilidade de aprender seu ambiente e melhorar seu desempenho, moldando-se ao padrão estipulado (HAYKIN, 2001). Esse padrão estipulado, para a RNA aprender seu ambiente, é determinado no momento de aprendizado da RNA.

A RNA é disposta de três camadas fundamentais, a camada de entrada a camada intermediária e a camada de saída. A camada intermediaria pode ser disposta por mais de uma camada, recebendo o nome de camadas ocultas, estas camadas interveem entre a entrada e saída da RNA de maneira eficaz, capaz de extrair estatísticas de ordem elevada conforme é adicionado mais camadas ocultas, fundamental quando o número de características na camada de entrada é alto.

As camadas ocultas e de saídas são compostas por neurônios artificiais, denominados nó, que fornecem um determinado valor na sua saída dependendo das informações vindas das entradas. A conexão entre esses nós se chama sinapse, sendo o peso sináptico a força entre essas conexões, que dá à RNA a capacidade de armazenar o conhecimento adquirido (HAYKIN, 2001). Desta forma cada nó de uma camada oculta, conecta-se com todos os nós da camada seguinte. A figura 3 demostra essa interação entre os nós, chamada sinapse.

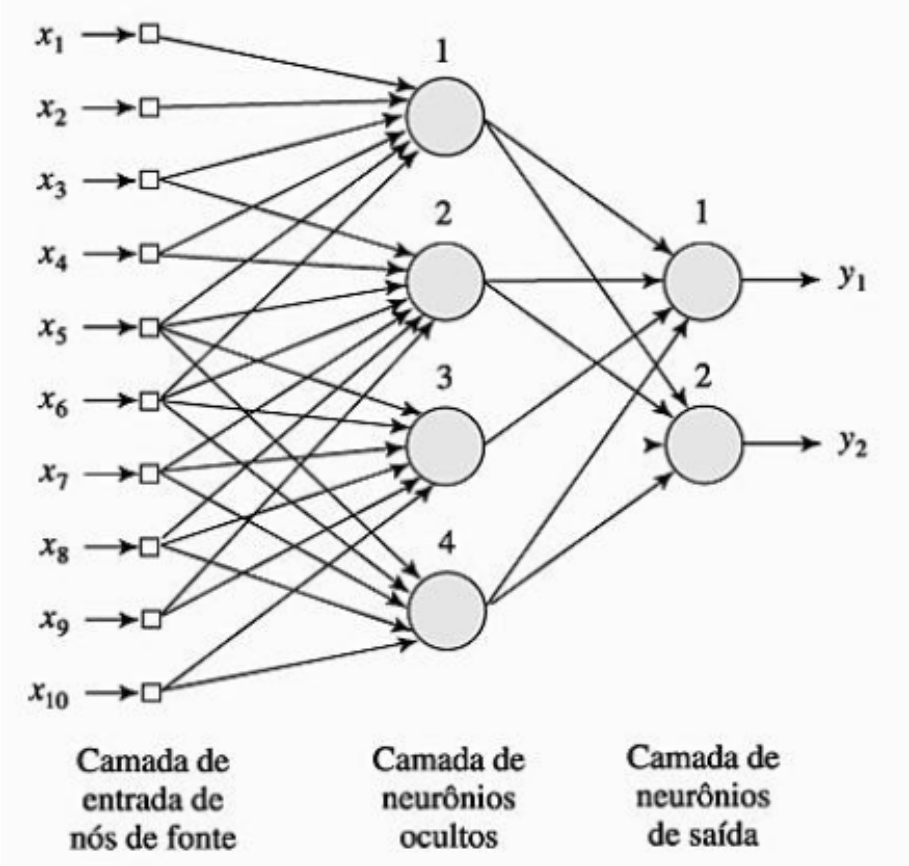


Figura 3 – RNA com dez dados na camada de entrada, 4 nós na camada oculta e 2 nós na camada de saída.

Fonte: (HAYKIN, 2001).

Há diversos tipos de estrutura para redes neurais, dentre elas, a perceptron de camada simples, perceptron de múltiplas camadas, adaline, redes recorrentes, entre outras.

2.3.1 Neurônio artificial

HAYKIN (2001) descreve que, o neurônio artificial é uma unidade de processamento de informações, definido por três elementos básicos, dentre eles; conjunto de sinapses, um somador e uma função de ativação. A Figura 4 esquematiza um neurônio artificial.

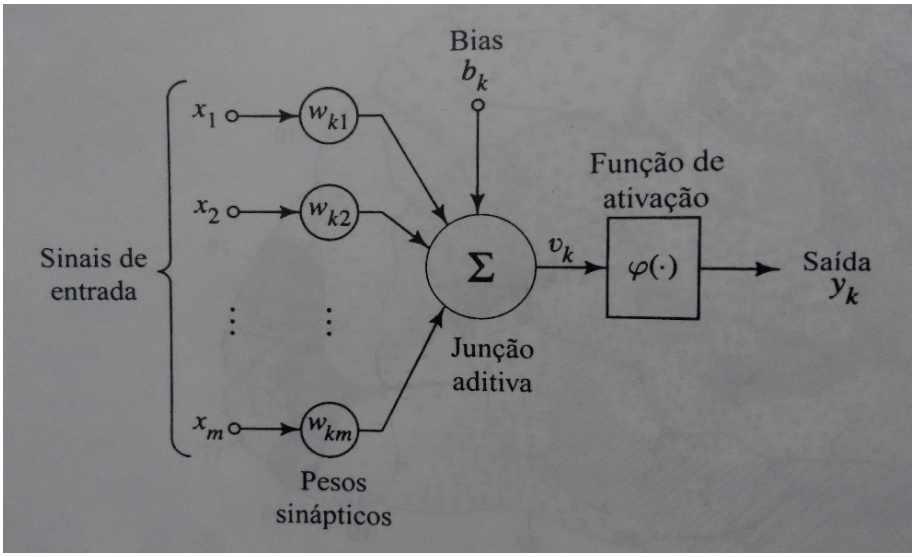


Figura 4 – Representação de um neurônio artificial.
Fonte: (HAYKIN, 2001).

Um conjunto de sinapses conectam as entradas da RNA ao somador, multiplicando os sinais de entrada ($x1$, $x2$) pelos pesos sinápticos ($Wk1$, $Wk2$), chegando até o somador, representado na figura 4 como junção aditiva. A junção aditiva soma os sinais oriundo de cada sinapse com um outro sinal chamado Bias, então o sinal de saída da junção aditiva, chamado de potencial de ativação, é excitado por uma função de ativação, que limita o valor de saída do neurônio, descrita como Yk na figura 4, entre um intervalo restrito. Normalmente esse intervalo é descrito como intervalo fechado [0,1], ou alternativamente [-1,1]. O bias tem o efeito de aumentar ou diminuir o potencial de ativação, dependendo se o bias tem um valor positivo ou negativo.

Existem três tipos básicos de função de ativação de acordo com haykin2001redes, representados na Figura 5. A função de limiar leva a saída do neurônio a zero caso o potencial de ativação seja menor que zero, e a um, caso o potencial de ativação seja maior ou igual a zero. A função liminar por partes, representada na figura 5, deixa a saída do neurônio em zero caso o potencial de ativação seja menor que -0,5, e deixa em um caso o potencial seja superior a 0,5. O tipo mais usado de função de ativação é a função sigmoide, pois é uma função estritamente crescente que exibe um balanceamento adequado entre comportamento linear e não linear, e também é uma função diferenciável, permitindo o cálculo do gradiente local, usado no algoritmo de retro propagação. A figura 5 esquematiza as principais funções de ativação.

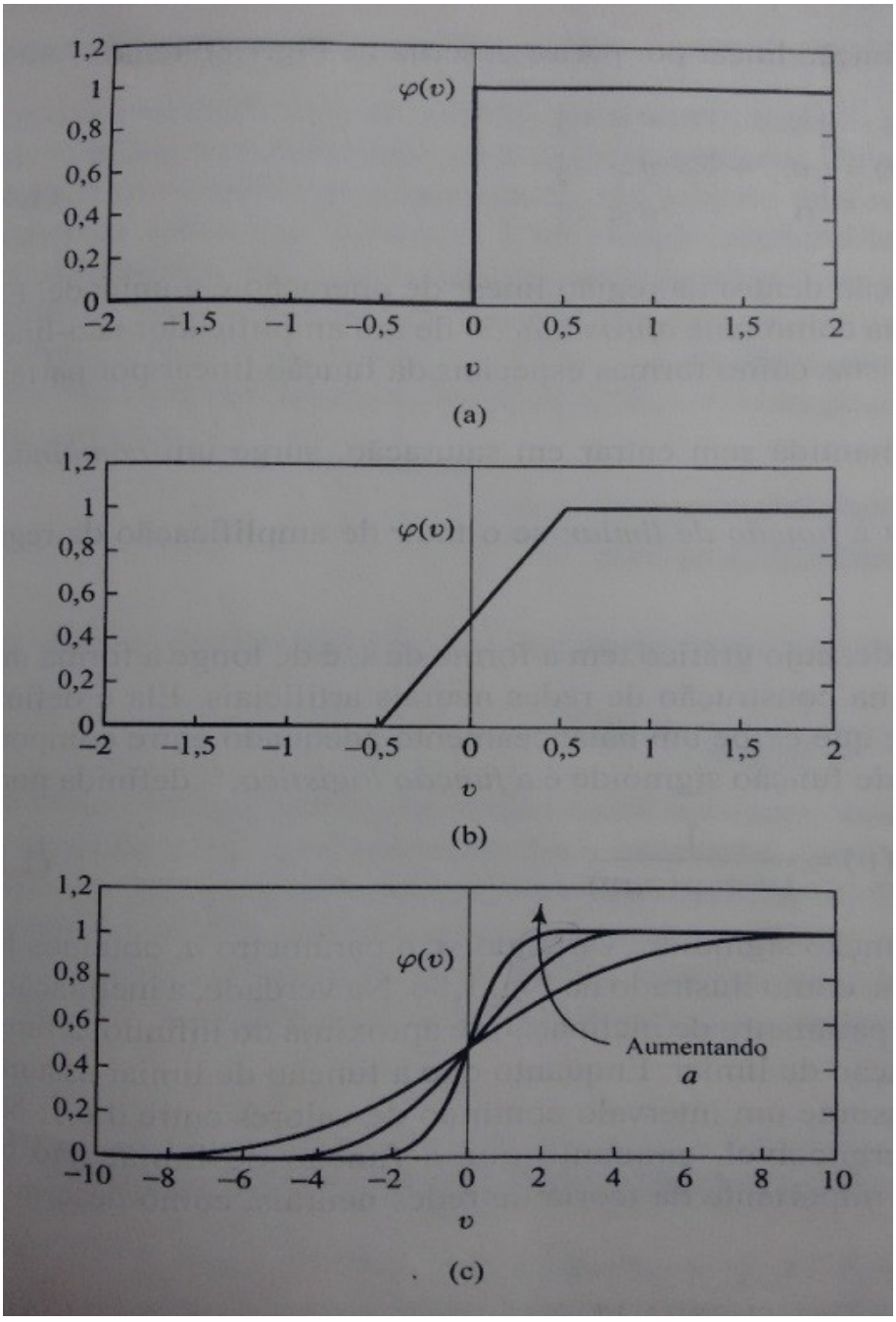


Figura 5 – (a) Função de Limiar. (b) Função Linear por Partes. (c) Função Sigmoide.
Fonte: (HAYKIN, 2001).

2.3.2 Rede perceptron de camada única

A rede perceptron de camada única é uma estrutura de RNA, que consegue separar suas classes linearmente como se fosse um plano, ou um hiperplano que contém as classes A e B, assumindo dois valores (Verdadeiro ou falso), onde é possível classificar os padrões mapeando a classe do problema, se os mesmos forem linearmente separáveis, assim representados por Minsky e Papert (1987).

A Perceptron é constituída basicamente de um único neurônio, com duas hipóteses na saída, verdadeiro ou falso. Aumentando o número de neurônios na camada de saída, é possível realizar a classificação com mais de duas classes, porém as classes devem ser linearmente separáveis para o funcionamento adequado do Perceptron, isso significa que os padrões a serem analisados não devem ser muito parecidos, para que o hiperplano seja uma linha reta, como na figura 6 A. Caso as duas classes estejam muito próximas (sejam muito semelhantes), como mostra a figura 6 B, elas se tornam não linearmente separáveis, pois terão algumas características em comum, e a perceptron de camada simples não terá a capacidade de distingui-las.

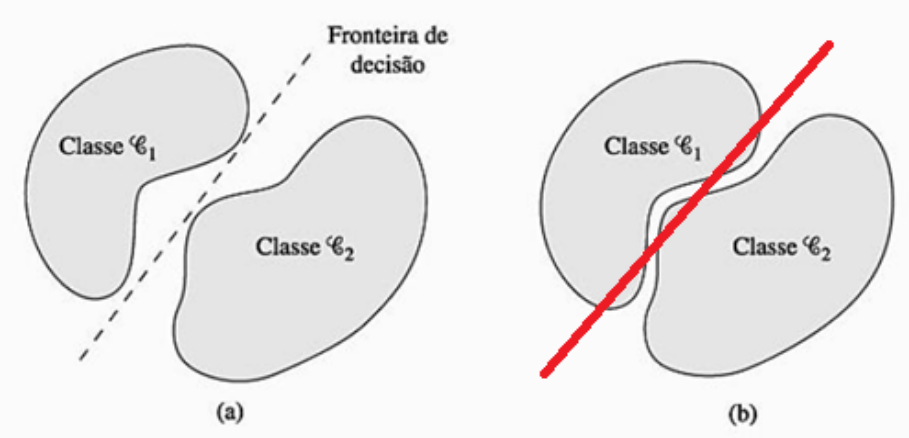


Figura 6 – (a) Classes linearmente separáveis. (b) Classes não linearmente separáveis.
Fonte: (HAYKIN, 2001), modificado.

O treinamento completo de um padrão, a partir de vetores de entrada na rede, é chamado de época, sendo esta responsável para que a inclinação do limiar de decisão entre as classes A e B, representada na figura 7, seja inclinado de maneira otimizada com o passar das épocas, através do ajuste dos pesos sinápticos, feito pelo método de correção de erro, que torna a separação entre as classes A e B bem definidas. Os valores de X1 e X2 da figura 7, referem-se a valores vindo da entrada da rede, que são ponderados pelos pesos sinápticos.

A distância entre o limiar de fronteira da figura 7, e a origem, está intimamente ligado ao valor do potencial de ativação do neurônio, portanto, o bias tem a função de aproximar ou afastar esse limiar da origem, sendo que, a linha da fronteira de decisão sempre passa pela origem quando o bias é zero.

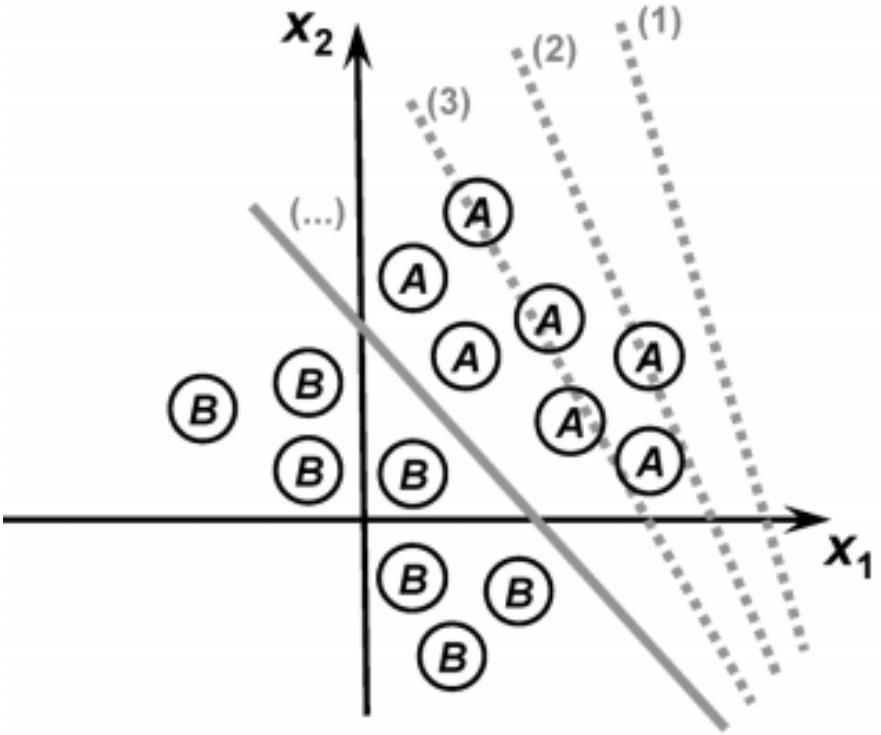


Figura 7 – Representação de duas classes linearmente separáveis.
Fonte:Carolina Yukari Veludo Watanabe

2.3.3 Rede perceptron de múltiplas camadas

Normalmente, a estrutura neural Perceptron de múltiplas camadas, possui um conjunto de unidades sensoriais, que são os dados fornecidos à camada de entrada, e também uma ou mais camadas ocultas, e uma camada de saída, propagando o sinal de entrada, camada por camada, até os neurônios de saída, de acordo com HAYKIN (2001). Analisando esta estrutura, observa-se que com mais camadas ocultas, é possível resolver problemas não lineares de maior dificuldade, onde possuímos um hiperplano com mais variáveis da camada de entrada, sendo possível separar mais de duas classes não lineares.

Através de algoritmo de treinamento supervisionada de retro propagação de erro, a rede Perceptron de múltiplas camadas tem sido utilizada para inúmeras aplicações, como por exemplo, o reconhecimento de caracteres, a previsão de comportamento na bolsa de valores, diagnósticos médicos, identificação e controle de processos, previsão de séries temporais, aproximação universal de funções, otimização de sistemas, e classificação de padrões, como por exemplo, reconhecimento de comando de voz, que possui um sistema mais complexo para identificação e qualidade vocal, através da amplitude, periodicidade e o tempo daquele sinal espectral.

A função de ativação usada em cada neurônio deve ser do tipo não linear, caso contrário a relação de entrada-saída da rede poderia ser reduzido a perceptron de camada única. HAYKIN

(2001) cita que, para aumentar a velocidade de aprendizagem do modelo perceptron de múltiplas camadas usando aprendizagem por retro-propagação, é aconselhável usar uma função de ativação antissimétrica, ou seja, normalizar a saída dos neurônios entre -1 e 1, usando uma função de ativação do tipo tangente hiperbólica.

2.3.4 Algoritmo de retropropagação (error back-propagation)

O algoritmo de retro propagação de erro é baseado no algoritmo de aprendizagem por correção de erro, sendo muito utilizado pela estrutura de RNA perceptron de múltiplas camadas, já que oferece um método computacional eficiente para o treinamento (HAYKIN, 2001).

De acordo com Braga, Carvalho, e Ludemir (2000), o treinamento desta estrutura de rede acontece em duas fases. A fase Forward, representada na Figura 8, é utilizada para gerar um conjunto de saídas reais, que devem ser a mais próxima possível da resposta desejada, produzindo um sinal de erro em cada neurônio da camada de saída. Nesta fase as informações vindas da entrada, chamada de sinais funcionais, se propagam camada a camada até a saída, mantendo fixo os pesos sinápticos.

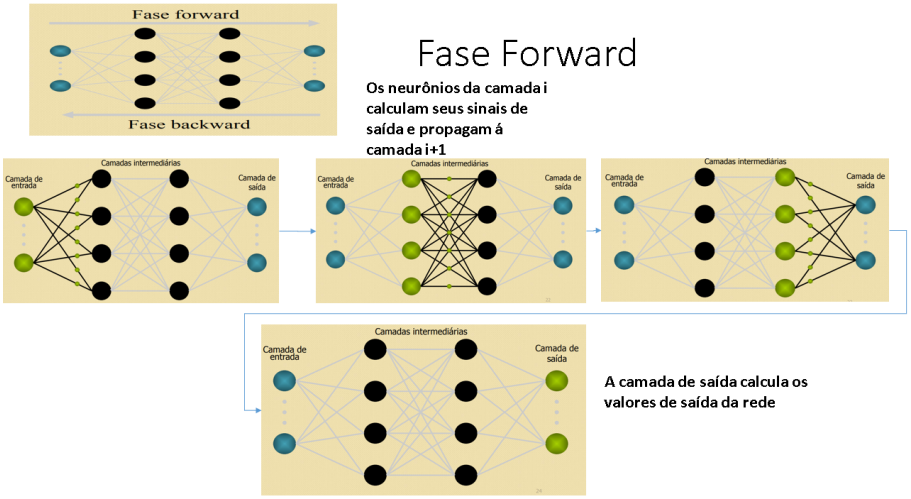


Figura 8 – Fase forward
Fonte:Centro de Informática(UFPE)

A Figura 9 esquematiza a fase Backward, esta fase propaga o sinal de erro a partir da saída até a entrada, camada por camada, ajustando os pesos sinápticos através de uma função que depende do sinal de erro, gerado em cada neurônio de saída na fase Forward, minimizando ao máximo o erro desejado com o treinamento da RNA.

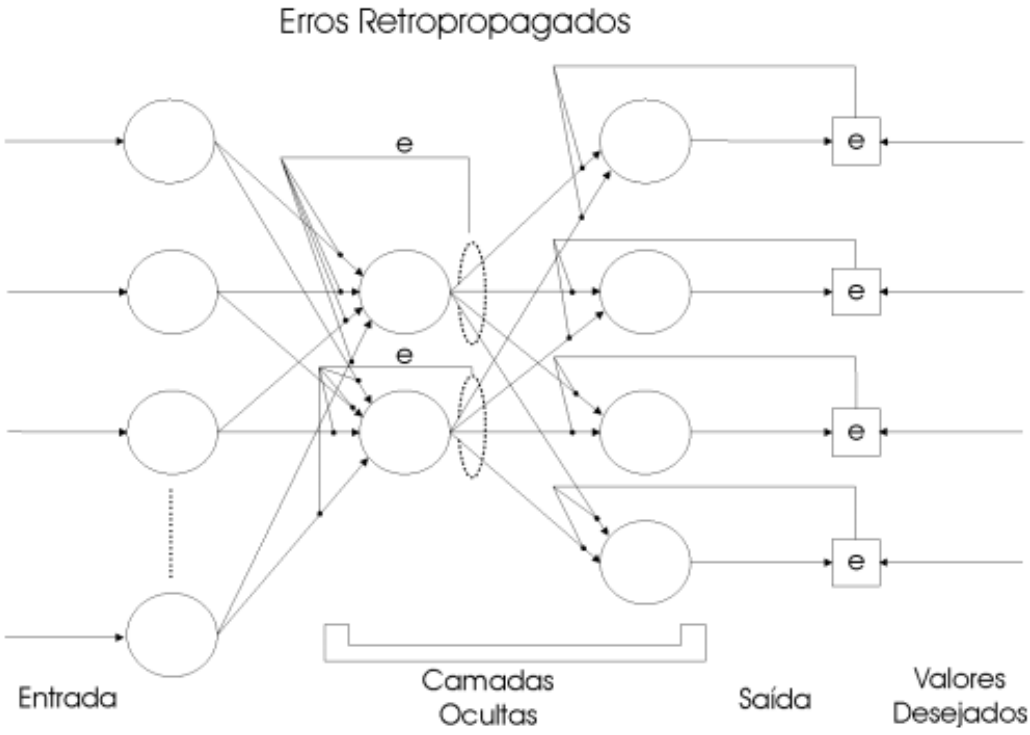


Figura 9 – Rede MLP com os acoplamentos retrógrados para os ajustes Sinápticos
Fonte:UFSC

2.4 SISTEMA AUDITIVO HUMANO

Nos seres humanos, conforme ALMEIDA (2014) descreve, as informações contidas nos sinais sonoros são captadas pelos ouvidos e transmitidas ao cérebro por meio de impulsos elétricos, para então serem interpretadas. Portanto, cabe ao ouvido humano, transformar a vibração mecânica proveniente de um sinal sonoro em impulsos nervosos, a fim de disponibilizar de forma eficiente a informação a ser interpretada e processada pelo cérebro.

O som é o movimento organizado das moléculas que compõem o meio, provocado pela variação brusca de posição de uma massa, como descreve Donoso (2018), que hospedava essas moléculas ao seu redor. Ao chegar nos ouvidos, o som vibra o tímpano, que está ligado a uma cadeia ossicular formada pelo martelo, bigorna e estribo, esses ossículos transmitem a vibração específica do sinal sonoro para dentro da cóclea, como mostra a figura 10.

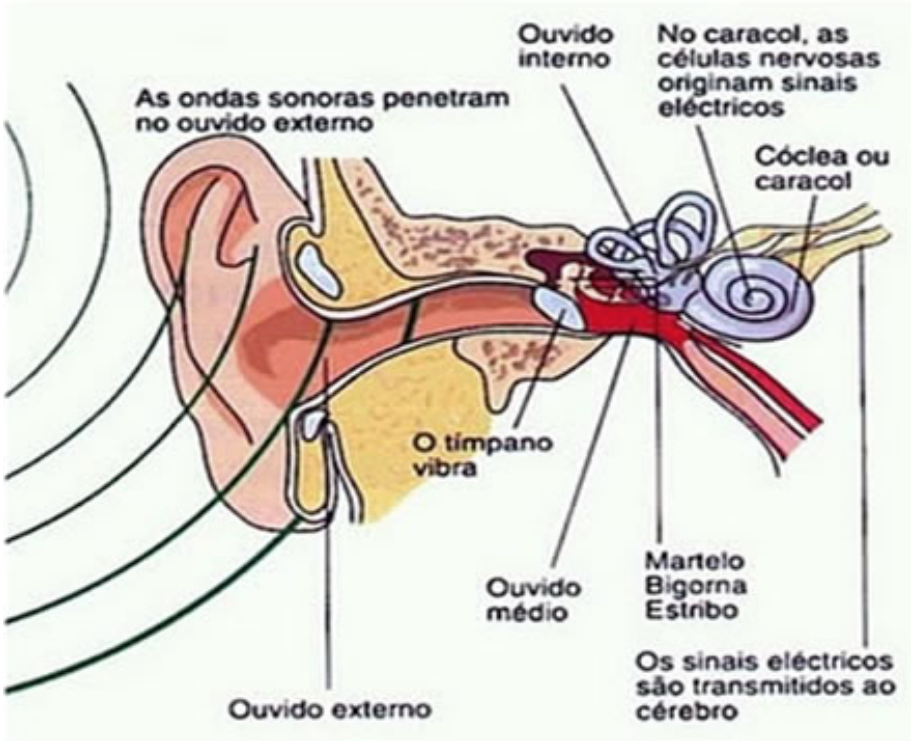


Figura 10 – Sistema auditivo humano.
Fonte:Mundo e educação, o ouvido humano.

É na cóclea, conhecida também como caracol, que o movimento mecânico se transforma em impulsos nervosos. A cóclea é uma estrutura cônica preenchida por líquido, por onde a vibração transmitida pelos ossículos se propaga, movimentando a membrana basilar, ilustrada na figura 9, que por sua vez faz as células ciliadas contraírem-se, gerando um diferencial de potencial elétrico nestas células, desencadeando uma corrente elétrica ao longo do nervo auditivo até o cérebro, onde a informação é processada.

O ducto Coclear, dispõem de células ciliadas ao longo de sua estrutura, que são excitadas de forma seletiva para cada frequência, sendo sensíveis a altas frequências no início do ducto, e sensível à baixas frequências na sua extremidade, como representado pela figura 11, funcionando com vários filtros passa faixa.

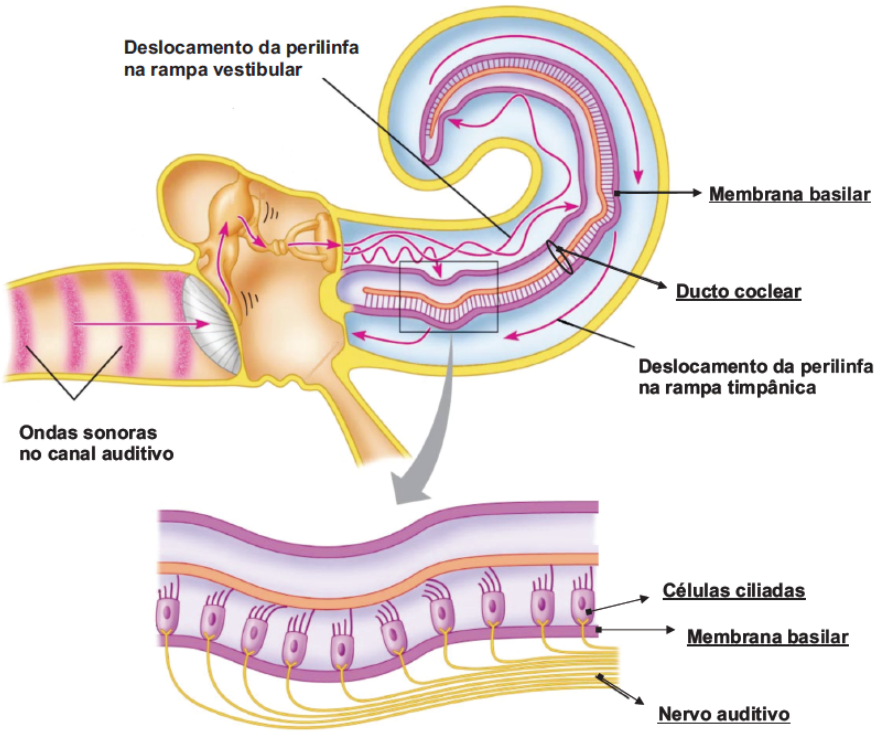


Figura 11 – Representação do ouvido interno.
Fonte:Droualb. Chapter 10: sensory.

O ouvido humano não consegue distinguir frequências que estão muito próximas, devido a interação entres células ciliadas vizinhas, limitando a percepção espectral, tal limitação é conhecida como banda crítica, Almeida (2014), e quanto maior é a frequência maior é a banda crítica, ou seja, nas altas frequências, as componentes harmônicas do sinal de alta frequência não podem ser percebidas separadamente. O ouvido consegue distinguir quando uma nota musical troca de 100 Hz para 200 Hz, mas não consegue distinguir quando a nota muda de 1100 Hz para 1200 Hz.

2.5 TRATO VOCAL HUMANO

O desenvolvimento humano passou por várias etapas desde os primórdios da humanidade. A comunicação, assim como o ser humano, também passou por processo evolutivo. O homem primitivo ao desenvolver a fala de forma conexa passa a se comunicar com outros de forma concreta, troca experiências e, desta forma, aprende coisas novas, o que permite a evolução.

A linguagem é vista como uma característica social fundamental da sociedade. Esse processo de falar não e tão simples como se mostra no dia-a-dia, podemos não perceber, mas a técnica desenvolvida pelo nosso corpo, a qual nos permite emitir som, e mais complexa do que parece.

O conjunto de órgãos responsáveis pela fala é conhecido como aparelho fonador. De acordo com Sègres (apud Jannibelli, 1971), a constituição do Aparelho Fonador se faz pelo Sistema Respiratório, Sistema de Emissão da fala, Sistema de Articulação e Ressonância, Sistema Nervoso Central e Periférico. Cada ser humano tem uma estrutura que permite características exclusivas relacionadas à voz. Essa estrutura é composta por órgãos conhecidos como ressoadores, para Amado (2014) "os órgãos ressoadores são, fundamentalmente, a faringe, o véu do palato, a boca e as fossas nasais. Cada uma destas estruturas apresenta-se com características específicas para o trato da voz." (p.50).

O ar ao passar pelo trato vocal produz a fala. almeida1, ressalta que o trato vocal vai desde a abertura das pregas vocais até os lábios, a qualidade do som produzido depende do tamanho da cavidade do trato vocal, onde o ar bate nas pregas vocais e fazem com que elas vibrarem várias vezes por segundo, produzindo uma certa frequência, que por sua vez gera ondas no espectro sonoro. Essas ondas geram uma amplitude maior onde o som será grave, atenuando conforme a frequência aumenta. Este sinal é constituído por amplitude, tempo e formas de ondas espectrais. A figura 12 esquematiza as principais formas do trato vocal e o padrão de saída de suas frequências de ressonância.

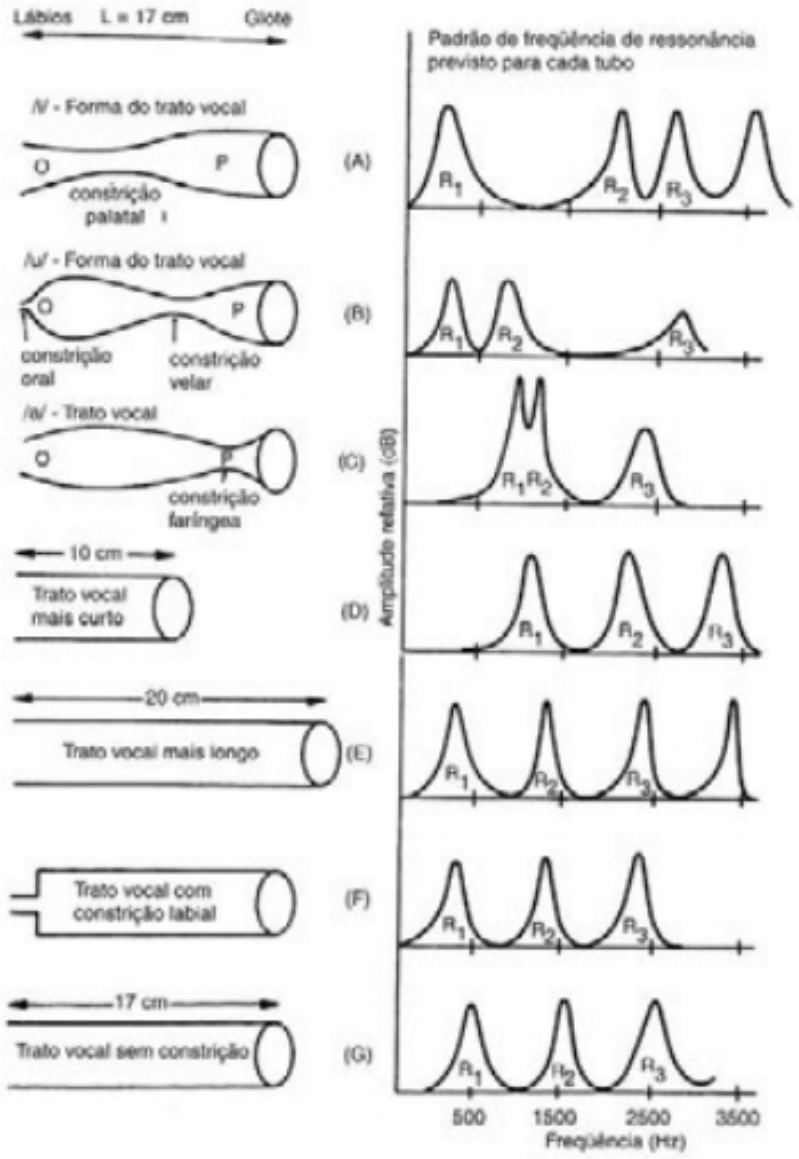


Figura 12 – Formação do trato vocal e suas formantes.
Fonte: GUSMÃO.

O espectro do sinal de voz para uma determinada palavra, nunca será o mesmo, por mais que o locutor se esforce para produzir o mesmo som. A palavra trato vem de tratar o que significa tomar nova feição, ganhar novo estado, característica, desta forma é tratada a frequência e suas formantes pelo trato vocal sendo ela fonema ou vogal, executando a função de filtragem das impurezas do sinal antes de passar pelos lábios.

2.6 MEL FREQUENCY CEPSTRAL COEFICIENTE (MFCC)

Para alimentar uma RNA para reconhecimento de comandos vocálicos adequadamente, é necessário descartar componentes irrelevantes daquele padrão vocálico analisado, como por

exemplo; ruído de fundo, emoção, gênero, entre outras características desnecessárias, tornando o processamento mais rápido e o reconhecimento mais eficiente. O MFCC é um método de extração de características de áudio, que está entre os mais utilizados.

O MFCC é uma técnica proposta por Mermelstein, na década de 70. Este método realiza a extração de características, denominadas componentes cepstrais, através da aplicação pela transformada dos cossenos no cálculo cepstral em intervalos de um sinal de áudio, separados por uma escala logarítmica melódica, denominada escala Mel (ZHENG F.; ZHANG, 2001).

A extração usando MFCC é baseada em duas características da audição humana segundo (ALMEIDA, 2014): a resposta em frequência da membrana basilar, onde existe uma banda crítica [2] é também a característica de excitações de compressão não lineares do nervo auditivo, geralmente é necessário 8 vezes mais energia para dobrar a intensidade de um sinal de áudio.

Para simular a resposta em frequência da membrana basilar, e da banda crítica de cada faixa de frequência, é usado um banco de filtros triangulares distribuído na escala Mel, com diferentes bandas de passagem, como demonstrado na figura 13. Conforme a frequência aumenta, maior é a banda de passagem.

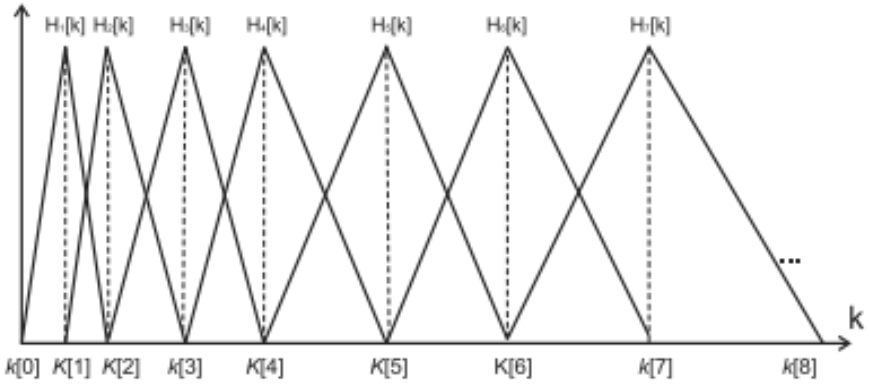


Figura 13 – Banco de filtros triangulares na escala Mel.
Fonte: PUC-Rio.

O método MFCC é aplicado a quadros que dividem o sinal de áudio, estes quadros podem ser sobrepostos ou não. Geralmente quando há sobreposição de quadros, é aplicado uma função de janelamento para atenuar as descontinuidades causadas no início e no final do sinal de cada quadro, e também para compensar a suposição feita pela Fast Fourier Transform (FFT) de que os dados são infinitos. Desses quadros são extraídas as componentes cepstrais, que contêm a soma da potência cepstral resultante relacionada a cada filtro triangular.

É importante que o quadro seja pequeno o suficiente para garantir que o sinal nesse intervalo seja estacionário, e grande o suficiente para obter uma estimativa espectral confiável. Para processamento de áudio é comum usar um quadro de 20 ms a 40 ms (FAYEK, 2016).

[2] Ver tópico: (2.4 Sistema auditivo Humano)

A lógica em dividir o sinal de áudio em sinais menores, é que as frequências de um sinal de áudio mudam com o tempo, e quando o sinal é analisado num curto período de tempo, a frequência do sinal pode ser considerada constante, então, na maioria dos casos, não faz sentido fazer a transformada de Fourier ao longo de todo o sinal. A transformada de Fourier é aplicada no sinal contido em um quadro, e então o módulo da transformada é filtrado pelo banco de filtros triangulares.

Para simular a característica de excitações de compressão não lineares do nervo auditivo, é calculado o logaritmo das componentes cepstrais e também a trasnformada discreta dos cossenos, como demostrado na figura 14.

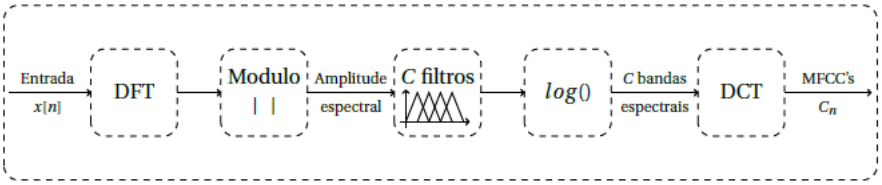


Figura 14 – Diagrama em bloco do processo de MFCC.
Fonte: (SILVA, 2015)

2.7 MESSAGE QUEUING TELEMETRY TRANSPORT(MQTT)

O Message Queuing Telemetry Transport (MQTT) foi desenvolvido pela IBM em 1999, sendo amplamente utilizado nos dias de hoje na Internet das coisas (IoT) e indústria 4.0. O MQTT é um protocolo de mensagens desenvolvido com base na pilha TCP/IP, com suporte para comunicação assíncrona entre as partes, sendo ideal para conexão de diversos dispositivos que se conectam entre si utilizando uma arquitetura de publicação(publisher) e assinatura(subscriber) segundo YUAN (2017). A publicação de uma informação, como por exemplo, a temperatura enviada a um Broker através de um tópico que gerencia a comunicação M2M (machine-to-machine) repassando as informações para os assinantes que se inscreveram no tópico correspondente, como na imagem 15.

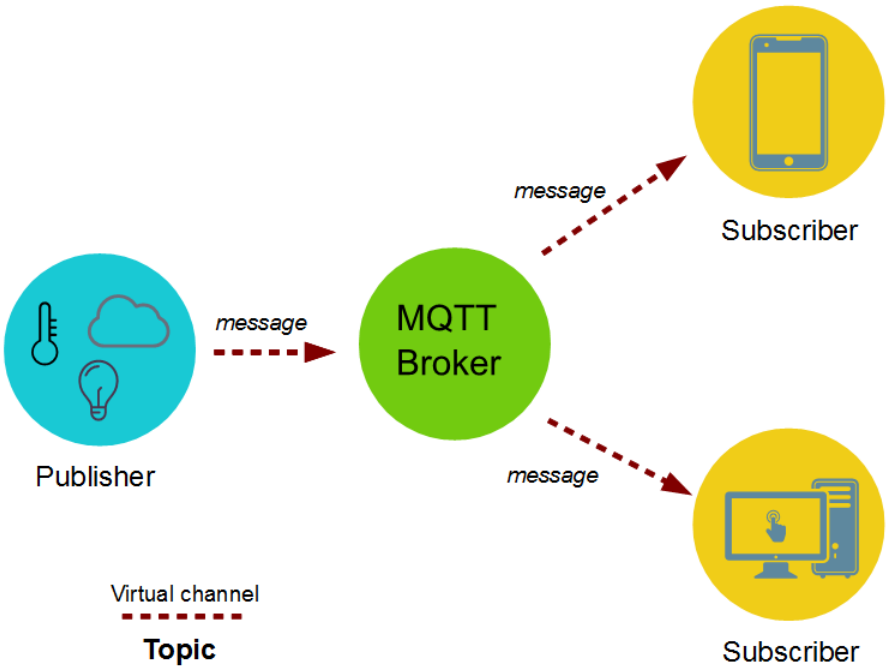


Figura 15 – Funcionamento do da comuniação MQTT
Fonte:1sheeld(2018)

3 DESENVOLVIMENTO

Este tópico apresenta detalhes sobre o desenvolvimento do protótipo, como por exemplo, o tratamento do sinal de áudio, a construção do banco de dados, o desenvolvimento dos códigos de captura de áudio, extração de coeficientes cepstrais, treinamento da RNA, e o código final de reconhecimento de comandos.

3.1 MICROCONTROLADOR

O microcontrolador escolhido para o desenvolvimento do projeto é a Orange PI Lite da Shenzhen Xunlong Softwares CO, Limited. Este microcontrolador possui uma arquitetura ARM Cortex A7, Quad-Core com 512MB DDR3.

As escolhas primordiais pela Orange Lite, foram o microfone embutido, sua antena wifi, e seu processador, aliando esses fatores ao custo benefício, dispensando a construção de um circuito para o microfone e a utilização de um adaptador wifi, focando no desenvolvimento dos códigos em linguagem python, operando todo programa em um sistema operacional com interface chamado Armbian Xenial legacy kernel 3.4y, uma distribuição leve baseada em Debian ou Ubuntu, construída especialmente para placas de desenvolvimento como a utilizada neste projeto, representada na figura 16.

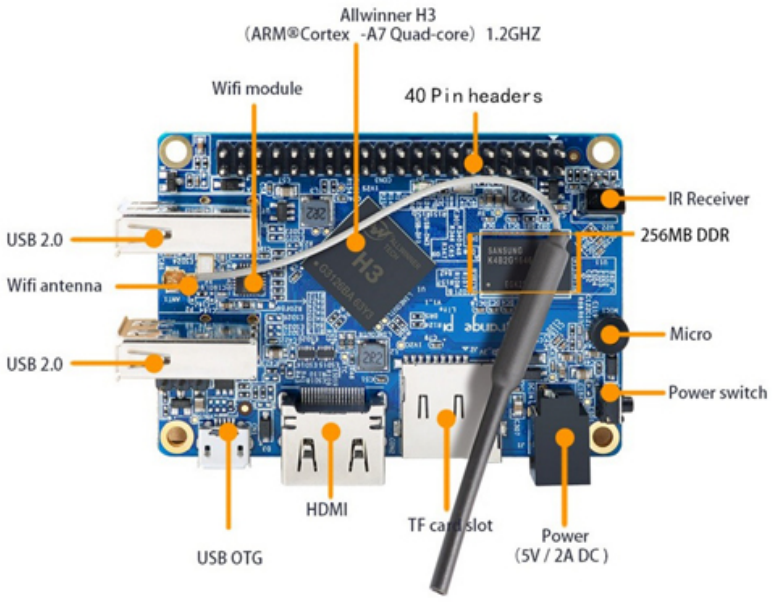


Figura 16 – Representação da placa prototipada.
Fonte:Orangepi.org(2018)

3.2 BANCO DE DADOS

O banco de dados foi construído a partir da voz de 62 voluntários, dentre eles, homens e mulheres fumantes e não fumantes, idosos e crianças. Sendo que cada voluntário gravou três vezes os seguintes comandos:

- Jarbas
- Ligue
- Desligue
- Música
- Sala
- Quarto
- Cozinha
- Ar-condicionado
- Tv
- Café

Para realizar a gravação dos voluntários, foi usado o código disposto no anexo A. Este código grava automaticamente um áudio de dois segundos para cada comando, com uma frequência de amostragem de 16000 bits por segundo.

Para acrescentar informações prévias e invariâncias às transformações do sinal de entrada, como sugerido no tópico 2.2, foi usado um programa para o Python, denominado Pysox, para criar novos arquivos de áudio com alteração na velocidade, de pitch.

A partir dos áudios originais, foram criados áudios 10% mais rápido, 20% mais rápido, 10 % mais lento e 20% mais lento, aumentando em cinco vezes o tamanho do banco de dados. Após a criação de áudios com alteração de velocidade, foram criados novos áudios com alteração no pitch, de um semitom mais grave e também de um semitom mais agudo, a partir dos áudios originais e também dos áudios com alteração de velocidade, aumentado em 15 vezes o tamanho do banco de dados.

3.3 TRATAMENTO DO SINAL DE ÁUDIO

Para poder gerar as componentes cepstrais dos áudios gravados pelos voluntários, através do MFCC, é necessário primeiro tratar o sinal de áudio, de forma adequada; eliminando as informações desnecessárias e não ter erros causado devido à resolução máxima do vetor discreto que carrega o sinal de áudio.

3.3.1 Pré ênfase

A pré-ênfase é usada para eliminar uma tendência espectral de aproximadamente -6dB/oitava na fala, irradiada dos lábios. Essa compensação na atenuação pode ser aplicada através de um filtro FIR de primeira ordem, de resposta aproximadamente +6dB/oitava, ocasionando um nivelamento no espectro. Enfatizando a porção do espectro mais distante da frequência fundamental. A imagem 17 mostra o comando de áudio "Jarbas"sem o filtro de pré-ênfase no primeiro gráfico, e o áudio filtrado no segundo gráfico.

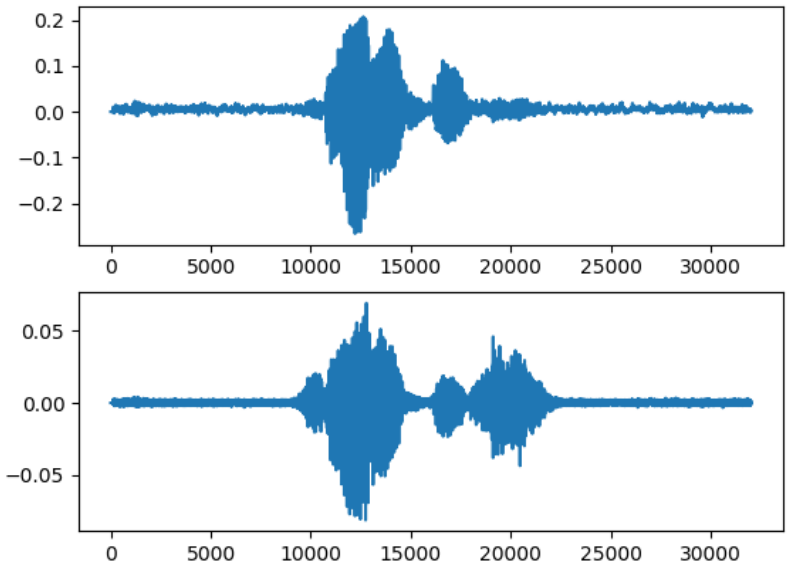


Figura 17 – No primeiro gráfico, o comando "Jarbas"sem o filtro de pré-ênfase. No segundo gráfico o comando está filtrado.
Fonte: Própria(2018).

3.3.2 Normallização

A normalização é um processo em que o sinal de áudio passa por um ajuste de amplitude, fixando um valor de ganho máximo pretendido, geralmente entre -1 e 1, evitando dessa forma que ocorra saturação das amostras, como descreve (NOCKO, 2006). No algoritmo escrito, o vetor que contém o áudio é analisado para que se tenha o valor de sua maior amostra como uma referência, então todas as amostras do vetor, incluindo aquela amostra de maior valor, são divididas por essa referência, fazendo com que o valor de pico seja 1, ou -1 caso a maior amplitude seja negativa.

3.3.3 Segmentação

Para simplificar o processamento da rede neural, foi desenvolvido um algoritmo de segmentação do sinal de áudio. A segmentação do áudio é utilizada para selecionar as principais informações do sinal, separando o comando dito, de regiões de silêncio, que não contém informação alguma sobre o comando de áudio, de acordo com Oppenheim e Schafer (1999), reduzindo quantidade de componentes cepstrais que serão processadas pela rede neural, otimizando o uso dos recursos computacionais e melhorando o desempenho do processo. Para isso, foi desenvolvido um algoritmo de segmentação baseado na energia contida no comando falado, já que a região não falada possui apenas ruído de fundo, em que a energia é muito baixa.

A energia de um sinal é amplamente utilizada em processamento de sinais, este conceito é demonstrado pela equação 3.3.1, que calcula a energia no tempo discreto (E[m]). Onde, $x[n]$ representa o valor contido na amostra n, i é a amostra do quadro, Q é a quantidade de amostras que compõem um quadro, p é o valor do passo, e m é o número do quadro que está calculando-se a energia MELO (2011).

$$E[m] = \sum_{n=i+(m*p)}^{Q+(m*p)} |x(n)|^2 (3.3.1)$$

A lógica por trás do algoritmo é detectar o início do comando a partir do limiar inferior de energia, que separa a região de silencio da região de fala. O limiar inferior de energia é uma proporção do pico de energia daquele comando de áudio. No projeto Jarbas, foi definido o limiar inferior de energia com um valor de 0,7% do valor da energia de pico daquele áudio. Para cada áudio analisado, o valor do limiar inferior de energia é alterado, dando dinâmica ao algoritmo de segmentação na hora de determinar o início do comando falado. Em ambientes ruidosos basta que o usuário fale mais alto, assim o limiar inferior de energia também aumenta.

Achado o início do comando falado, o algoritmo determina o final do comando após 17600 amostras. Desse modo a segmentação sempre retornará a mesma quantidade de amostras a serem filtradas pelo MFCC. Outro motivo pela utilização de segmentação por janela fixa, foi a agilidade no processamento de áudio quando comparado à segmentação por janela variável, onde o algoritmo determina o final do segmento usando o mesmo limiar inferior de energia. Nos testes realizados, a segmentação com janela fixa e janela variável tiveram a mesma taxa de acerto.

A energia do áudio é analisada a partir de um número determinado de amostras, essas amostras determinam um quadro, como demonstrado na figura 18, assim é possível calcular a energia de pequenos trechos do áudio e comparar com o valor do limiar inferior de energia. Caso o quadro analisado seja menor do que o liminar de energia, é provável que este faça parte de uma região de silêncio, se isto for verdade, o algorítimo marca como início do comando falado.

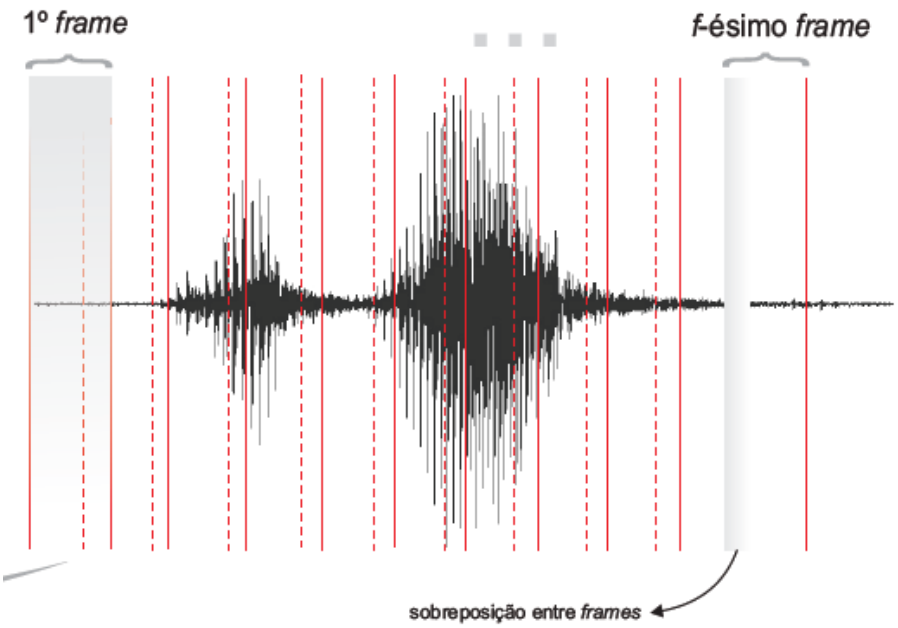


Figura 18 – Sinal de áudio dividido em quadros sobrepostos.
Fonte:(ALMEIDA, 2014).

Este método possui uma falha quando é usado para segmentação de comandos muito longos, com o "Ar-condicionado"por exemplo. Na figura 19, que mostra na parte superior, o gráfico de energia do comando de áudio "Ar-condicionado", fica claro que há várias regiões de vale, em que a energia possivelmente seja menor do que o limiar inferior, que é uma pequena região de silêncio, mas que faz parte do comando falado. No terceiro gráfico é possível ver que o algoritmo segmentou o comando do gráfico 2 pela metade, pois identificou uma falsa região de silêncio. O quarto gráfico apresenta o comando já segmentado. Para a segmentação da figura 19, foram usadas 800 amostras para compor um quadro, com sobreposição de 100 amostras.

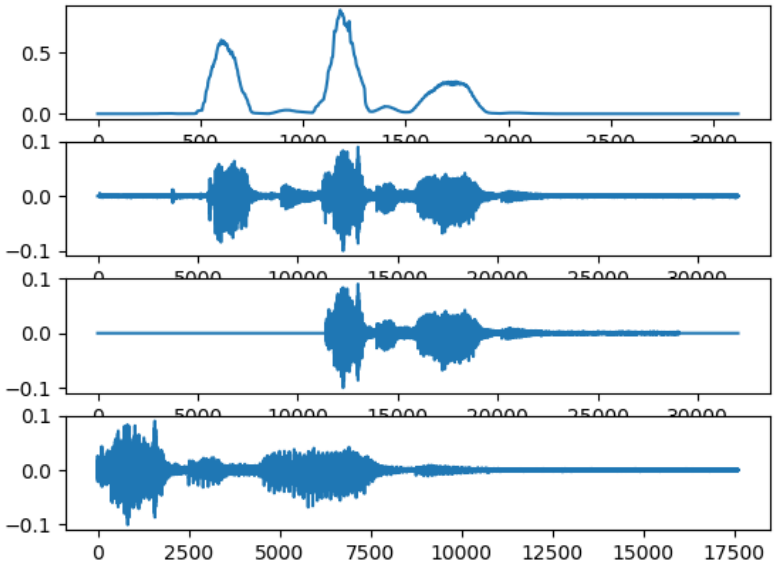


Figura 19 – Gráfico da energia e segmentação do áudio
Fonte:Própria(2018).

Para contornar o problema, foram usadas 3200 amostras em cada quadro, com sobreposição de 800 amostras. Dessa forma o gráfico de energia do sinal de áudio ficou com menor desvio padrão em relação à média de energia do sinal como um todo, em outras palavras, os picos e vales, que estão mais afastados do valor médio da energia, ficaram mais próximos da média, atenuando os picos, e amplificando os vales, como demostrado na figura 20. Essa estratégia deixa o sinal menos fiel, mas como o objetivo é analisar apenas a transição do valor de energia da região de silêncio para a região falada, não trouxe nenhum resultado negativo. Na figura 20 é possível verificar que o comando falado foi segmentado adequadamente.

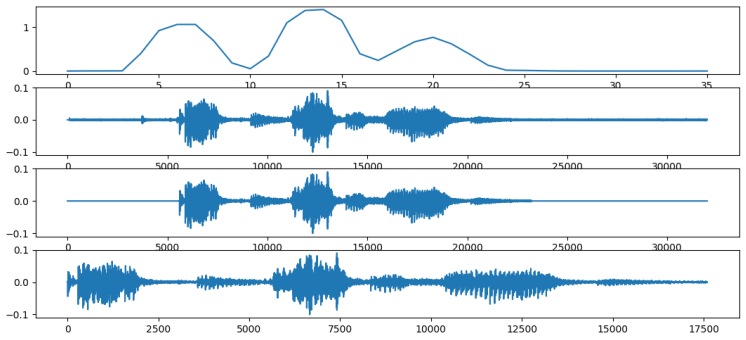


Figura 20 – Energia do áudio Ar-condicionado e segmentação do comando falado
Fonte: Própria(2018).

Outro benefício da estratégia utilizada, foi em relação ao processamento. Com um número maior de amostras compondo os quadros, o tamanho do vetor que receberá os valores da

energia de cada quadro diminuiu. Para a segmentação do banco de dados, com 32000 amostras em cada áudio, o vetor contendo a anergia de cada quadro da imagem 19, ficou com 312 amostras. Já na segmentação da figura 20, o vetor de energia ficou com 36 amostras, agilizando o processamento na hora de comparar cada amostra do vetor com o limiar inferior de energia.

Um áudio de dois segundos, gravado a uma frequência de amostragem de 16000 Hz, possui 32000 amostras. Com o algoritmo de segmentação, a quantidade de amostras que foram filtradas pelo MFCC diminuiu para 17600, equivalente a 1,1 segundos.

Para o algorítimo segmentar adequadamente, é importante que o limiar inferior de energia seja maior do que a energia contida nos quadros da região de silêncio do áudio, caso contrários o algoritmo entenderá que a região de silêncio faz parte da região falada.

No anexo B, está contido o código de segmentação utilizado para segmentar as regiões de fala das, regiões de silêncio, dos áudios do banco de dados.

3.4 UTILIZAÇÃO DO DESCRITOR DE CARACTERÍSTICAS MEL FREQUECY CEPSTRAL COEFICIENTE (MFCC)

Na etapa de extração das componentes cepstrais, do banco de dados segmentado, foi utilizado a biblioteca python speech features. Esta biblioteca fornece recursos comuns para reconhecimento de fala, como pro exemplo, MFCCs, energia e logaritmo de banco de filtros mel, centrômero de Sub-bandas espectrais (GITHUB, 2018).

O algoritmo responsável por extrair as componentes cepstrais do banco de dados, incia lendo o arquivo wav para um vetor e então normaliza este vetor entre -1 e 1, após isso, aplica-se ao vetor um filtro de pré-ênfase, e então é usado o algoritmo de segmentação para gerar um novo vetor com as amostras do áudio já segmentado, contendo apenas a região falada. Com o áudio segmentado, o sinal sofre uma normalização para deixar o áudio entre 1 e -1. O áudio normalizado é então filtrado pelo MFCC, e dessa filtragem, são extraídas 130 amostras de um vetor que continha 17600 amostras. Essas 130 amostras são as componentes cepstrais, que contém informação sobre a energia encontrada nas frequências que compõem aquele sinal de áudio.

A biblioteca Python Speech Feature, possibilita a configuração de alguns parâmetros, como pro exemplo, a quantidade de quadros que o sinal será dividido, a quantidade de filtros triangulares contidas em cada quadro, quais dos filtros serão usados, a frequência mínima e máxima a ser filtrada, e a pré-ênfase, que não foi habilitada porque o sinal já havia passado pelo filtro anteriormente.

Na filtragem da características cepstrais, o áudio segmentado pode ser divido em pedaços menores, chamados de quadro, desde que a quantidade de quadros seja um número par (CRYPTOGRAPHY, 2018). No projeto, esse áudio foi dividido em dez quadros sem sobreposição, cada quadro com 17600 amostras. Essas amostras passam por todos os filtros distribuídos

na escala mel, e cada filtro retorna uma componente cepstral. A biblioteca está configurada para usar apenas os 13 primeiros filtros, retornando 13 componentes cepstrais de cada quadro, totalizando 130 componentes para cada comando filtrado. O primeiro componente cepstral de cada quadro foi substituída pela logaritmo da energia contida nele.

No anexo C, está o algorítimo desenvolvido em python para a extração das características cepstrais. Para gerar um arquivo Comma Sepated Values (CSV) com as componentes cepstrais, basta adicionar o comando '» arquivo.csv' após o comando de excussão do algorítimo no terminal do Linux, por exemplo, $python mfcc.py » arquivo.csv. "Mfcc.py"é o nome do algoritmo, e "arquivo.csv"é o arquivo a ser gerado com o nome "arquivo".

3.5 COMMA SEPATED VALUES (CSV) E XTENSIBLE MARKUP LANGUAGE(XML)

O CSV é um arquivo de valores separados por vírgulas e agrupados em uma planilha, que contém componentes das amostras cepstrais que caracterizam o sinal de áudio, após a filtragem do MFCC, como por exemplo, "Jarbas" ou "ligue". Esses valores quantificados e tabelados foram utilizados para o treinamento da RNA, gerando um arquivo XML, que contém os valores dos parâmetros livres após o treinamento da rede neural. Deste modo o processo de treinamento, que exigem maior poder computacional, pode ser realizado em um desktop, otimizando o tempo gasto para treinamento da rede. Com o arquivo XML pronto, basta passar o arquivo para a placa Orange Pi através do usb, ou da internet.

3.6 TREINAMENTO DA RNA

Neste tópico será discutido como foi desenvolvido todo o processo para a implementação e treinamento da RNA do protótipo "Jarbas", por exemplo, as bibliotecas utilizadas, o método de treinamento, descritores selecionados e pôr fim a rede treinada.

3.6.1 Biblioteca Pybrain

Segundo (SCHAUL *et al.*, 2010) a biblioteca Pybrain é uma das poucas ferramentas que utiliza algoritmos para redes neurais em python, sendo construída em torno de redes neurais no kernel. Todos os métodos de treinamento aceitam uma rede neural como a instância a ser treinada, isso faz do PyBrain uma ferramenta poderosa para tarefas da vida real, podendo ser treinado diferentes tipos de aprendizado de máquina sendo ele supervisionado, por reforço, não – supervisionado, recorrente, entre outros. As bibliotecas usadas para treinamento da rede estão citadas a baixo;

- from pybrain.tools.shortcuts import buildNetwork;
- from pybrain.datasets import SupervisedDataSet;
- from pybrain.supervised.trainers import BackpropTrainer;
- from pybrain.structure.modules import SoftmaxLayer, SigmoidLayer;
- from pybrain.tools.xml import NetworkWriter, NetworkReader.

3.6.2 Método utilizado para treinamento da RNA

O método utilizado para treinamento da rede neural artificial foi o treinamento supervisionado com backpropagation, o mesmo discutido no capitulo 2.3.4, sobre treinamento por correção de erros.

A rede neural foi construída com 130 neurônios na camada de entrada, 36 na camada oculta, e 10 na camada de saída. Para um treinamento mais rápido foi ativado o "Bias", apresentando no tópico 2.3.1. Apesar das componentes cepstrais possuírem valores positivos e negativos, a função de ativação tangente hiperbólica para a camada oculta não foi a melhor escolha, mostrando-se pior que a função sigmoid squashing. Para a camada de saída foi usada a função de ativação softmax, por ser um problema que contém classificação.

O algoritmo usado para treinar a rede neural, está disposto no anexo D. Esse algoritmo importa o arquivo CSV, normaliza suas componentes entre -1 e 1, treina a rede neural, e por fim gera um arquivo XML, com o valor dos parâmetros livres da RNA.

3.7 MOSQUITTO

O protocolo utilizado foi o Eclipse Mosquito, que segundo (LIGHT., 2013), é um broker MQTT de distribuição livre, utilizado desde computadores de mesa única de baixa potência, até servidores complexos. Este protocolo foi utilizado na OrangePi Lite configurando o "mosquito"no microcontrolador como o emissor da informação e broker ao mesmo tempo, dispensando a necessidade de um servidor conectado à internet. Desta forma as informações foram enviadas para um aplicativo de dispositivo móvel, no caso My MQTT. Assim o usuário ou Publisher envia duas palavras para o broker, como por exemplo; TV-Ligue, ou Música-Desligue, que chega pelo aplicado como o assinante(subscriber), que está inscrito no broker chamado Jarbas.

3.8 JARBAS

Jarbas foi o nome escolhido para o protótipo por lembrar Jarvis, o assistente virtual do homem de ferro, um super-herói da Marvel Comics, assumindo o papel de um assistente residencial para pessoas idosas e com deficiência. Jarbas é uma rede neural que reconhece os padrões, e envia esses comandos via wifi. Ele é composto por uma case impressa na impressora 3D para abrigar a Orange Pi e inserido um led rgb difuso para interagir com o usuário, como representado pela figura 21.

Figura 21 – Jarbas de perfil.
Fonte: Própria(2018).

O funcionamento do jarbas se faz da seguinte forma: ao executar o algoritmo, a placa é acionada no estado de hibernação e o led acende na cor azul, ficando neste estado até que detecte em seu microfone uma energia maior do que um limiar definido, assim o áudio será gravado apenas quando for dito num volume alto, ou dito perto do microfone, evitando que o microfone fique gravando toda hora e desperdiçando energia. A medição do limiar de energia do microfone, possibilitou que o algoritmo final funcionasse sem o algoritmo de segmentação de áudio, pois o inicio da gravação também é o início do comando falado.

Se o comando for dito alto o suficiente para ser gravado, o algoritmo grava um áudio de 1,1 segundos e extrai as componentes cepstrais, alimentando a RNA. Esse processo pode ser chamado de processamento e nesse estado, o led acende a cor branca. Caso a rede entenda que o comando foi Jarbas, o algoritmo "acorda"da hibernação e o led acende na cor verde, caso contrário o led piscará na cor vermelha e voltará para o estado de hibernação.

O led verde indica que Jarbas está “acordado” e pronto para gravar os comandos, sendo o processo de gravação dessa etapa semelhante à etapa onde o Jarbas estava hibernando. Quando o primeiro comando, relacionado ao eletrodoméstico for dito alto o suficiente, será gravado um

áudio de 1,1 segundos e então o led piscará na cor verde duas vezes, indicando ao usuário que o segundo comando (Ligue ou Desligue) já pode ser dito. Quando o segundo comando for gravado, o led acende na cor branca, indicando que está ocorrendo o processamento.

A RNA processará os dois comandos, caso o último comando seja interpretado como Ligue ou Desligue, é enviado um pacote via MQTT com os comandos ditos, primeiro o usuário fala “Jarbas", após o led verde acender, ele fala Ar-condicionado e por último, ligue, após o led verde piscar duas vezes, assim enviando via wifi(MQTT) o comando Ar-condicionado-Ligue, então o protótipo voltará a hibernar. Caso o último comando não seja interpretado como "Ligue ou Desligue", ele ira piscar o led vermelho e voltará para a hibernação, neste caso não é enviado sinal algum de wifi.

Foi escolhido 1,1 segundos como tempo de gravação por causa do comando "Ar-condicionado", que possui maior duração, aproximadamente 1 segundo. O led branco indicando processamento é importante para que o usuário saiba que falou alto o suficiente para que a sua voz seja gravada. O código escrito em python está disposto no anexo E.

4 EXPERIMENTOS E RESULTADOS OBTIDOS

No tópico em questão, serão comentados o os testes realizados durante o processo de aprendizagem da RNA, como o número de épocas, função de ativação da RNA, o valor da taxa de aprendizagem, número de neurônios em cada camada, teste de distância e normalização dos dados apresentados à RNA.

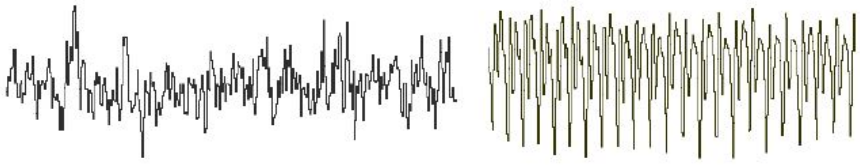
4.1 CRUZAMENTO POR ZERO

Além da segmentação usando energia, foi testado também a segmentação que, além da calcular a energia, verifica também a taxa de cruzamento por zero que ocorreu naquele quadro analisado.

A taxa de cruzamento por zero se trata de calcular a quantidade de vezes que o sinal cruza a média da amplitude normalizada, no quadro analisado, (MELO, 2011). No caso do projeto, o áudio estava normalizado entre -1 e 1, com uma média em zero.

O algoritmo multiplica o valor da amostra atual pela amostra anterior, e verifica se o resultado é menor que zero, caso afirmativo, isso significa que o sinal cruzou por zero. Isto é feito quadro por quadro, assim como é feito para calcular a energia. Tendo a quantidade de cruzamento por zero de cada quadro, é definido um limiar de cruzamento por zero, parecido com o limiar inferior de energia. No projeto foi testado um limiar de cruzamento por zero de 75% do valor do primeiro quadro.

Caso o valor da quantidade de cruzamento por zero do quadro seja maior que o liminar definido, a região possivelmente faz parte da região de silêncio. Isso porque regiões não vozeada, possuem uma taxa de cruzamento por zero bem maior do que região vozeada, como demostrado na imagem 23. O som vozeado é reproduzido pela vibração das cordas vocais, já no som não vozeado as cordas vocais não vibram.



(a) elocução do fonema /s/: Som não-vozeado. (b) Fonema /z/: Som vozeado.

Figura 22 – Análise da taxa de cruzamento por zero da região de silêncio representada na figura a), e região de fala na figura b).
Fonte: (MELO, 2011).

Após identificar o inicio da região de fala usando a comparação de energia, o algoritmo analisa a taxa de cruzamento por zero para verificar o início do comando falado.

Esse método se mostrou eficaz para a segmentação de alguns comandos, mas não para a segmentação de "Jarbas", pois no meio do comando há um trecho não vozeado, que é causado pelo som de "s", sem vibração das cordas vocais. Desse modo o método que analisa a taxa de cruzamento por zero foi descartado do algoritmos de segmentação.

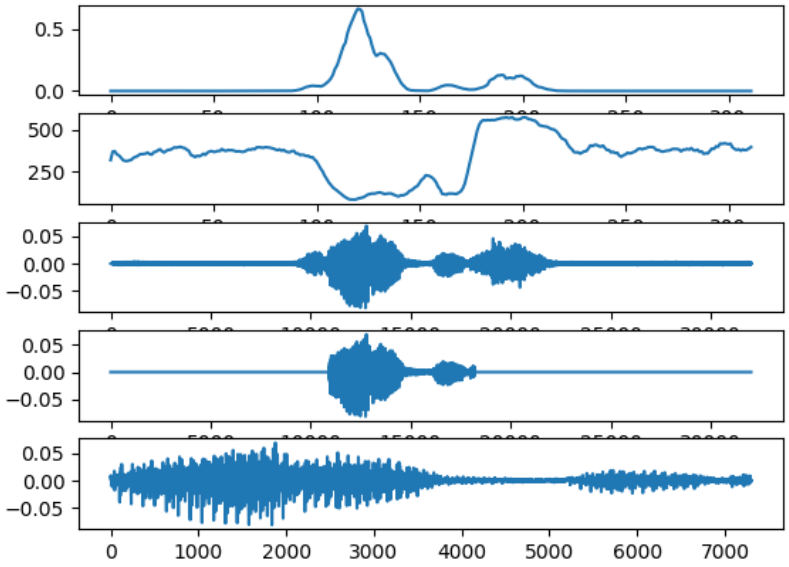


Figura 23 – No primeiro gráfico está representada a curva de energia do comando "Jarbas", e no segundo a curva da taxa de cruzamento por zero. O gráfico 3 mostra o sinal de áudio do comando "Jarbas". O gráfico 4 representa o vetor que zera os quadros entendidas como região de silêncio. O último gráfico representa somente a parte útil do comando de voz a parte segmentada.
Fonte: Própria(2018).

4.2 JANELA FIXA VS JANELA VARIÁVEL)

A segmentação por janela fixa entrega sempre a mesma quantidade de amostras para ao filtro MFCC, já a segmentação com janela móvel, após definir o início da área falada, procura o final. Os comandos segmentados no protótipo foram realizados com janela fixa, pois mostrou resultados semelhantes à segmentação por janela dinâmica.

Nos testes realizados, a taxa de acerto dos algoritmos com e sem janela fixa, apresentaram resultados semelhantes. Como o uso de janela variável precisa de um algoritmo de segmentação para achar o final da área de fala, o uso de janela variável foi descartado do projeto, otimizando o tempo de processamento, pois com a janela fixa, o início do comando falado é o início da gravação, recurso proporcionado pelo limiar de energia do microfone, e o final da área falada será definida como 1,1 segundos após o início.

4.3 TREINAMENTO DA RNA

Durante o treinamento da RNA foram testadas diversas características e métodos, a fim de procurar os melhores resultados.

As primeiras observações feitas, foram relacionadas ao *overfitting*. O *overfitting* ocorre quando há um sobre-ajuste dos parâmetros livres da RNA, caso a rede seja treinada com uma quantidade exagerada de épocas, a rede aprenderá com maior precisão os parâmetros daquele padrão treinado, podendo após o treinamento, não reconhecer o mesmo padrão de fala, caso algumas de suas características estejam um pouco diferentes daquelas usadas para o treinamento. Se a rede for treinada com excesso de épocas, ela poderá não distinguir adequadamente os comandos falados por pessoas que não participaram da construção do banco de dados, usado para o treinamento. O *overfitting* está intimamente relacionado ao número de épocas usadas para o treinamento da RNA.

Após a etapa de extração das características de áudio, utilizando MFCC, o código de treinamento da RNA, comentado no tópico 3.6.2, importa o arquivo CSV com as componentes cepstrais, e enquanto treina a rede, imprime o erro dos neurônios de saída na tela, época após época. Desta forma é possível analisar o erro. Quanto menor for o erro maior é a probabilidade da rede estar com *overfitting*.

Outra ferramenta que auxiliou para verificar o *overfitting*, foi a matriz confusão. Além de mostrar se a rede está com *overfitting*, a matriz informa a taxa de acerto após o treinamento. A matriz confusão é gerada a partir dos resultados obtidos e dos resultados esperados. Na imagem 24 as linhas representam os comandos, a linha 0 representa o comando "Jarbas", a linha 1 representa o comando "Ligue", a 9 representa "Café", e assim por diante. As colunas também representam os mesmos comandos, a coluna 0 representa "Jarbas", a 1 representa "Ligue", até a coluna 9, representando "Café", porém as linhas estão relacionadas aos comandos esperados, e as colunas à quantidade de comandos entendidos pela rede neural. Se a rede foi alimentada com 10 comandos "Jarbas", e dos 10 áudios, a rna entendeu 8 comandos sendo Jarbas, um sendo Ligue e o ultimo como Desligue, a primeira linha será apresentada como a baixo;

[8 , 1 , 1 , 0 , 0 , 0 , 0 , 0 , 0 , 0]

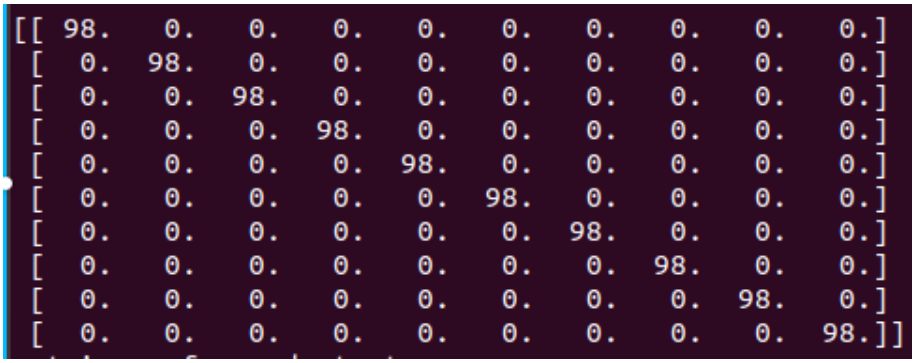


Figura 24 – Matriz confusão para os dez comandos de áudio
Fonte: Própria(2018).

A figura 25 mostra uma rna com overfitting, após a última época o erro está muito baixo, e a matriz confusão de treino mostra que a rede está acertando todos os áudios usados para o treinamento. A matriz confusão de treino foi gerada a partir dos áudios usados para o treinamento da rna, já a matriz confusão de teste foi gerada a partir de áudios que não foram usados para o treinamento da rna, esse são desconhecidos da rede. Para o treinamento da rna da imagem 25 foram usadas 400 épocas e uma taxa de aprendizagem de 0,08 com uma taxa de acerto foi de 80%.

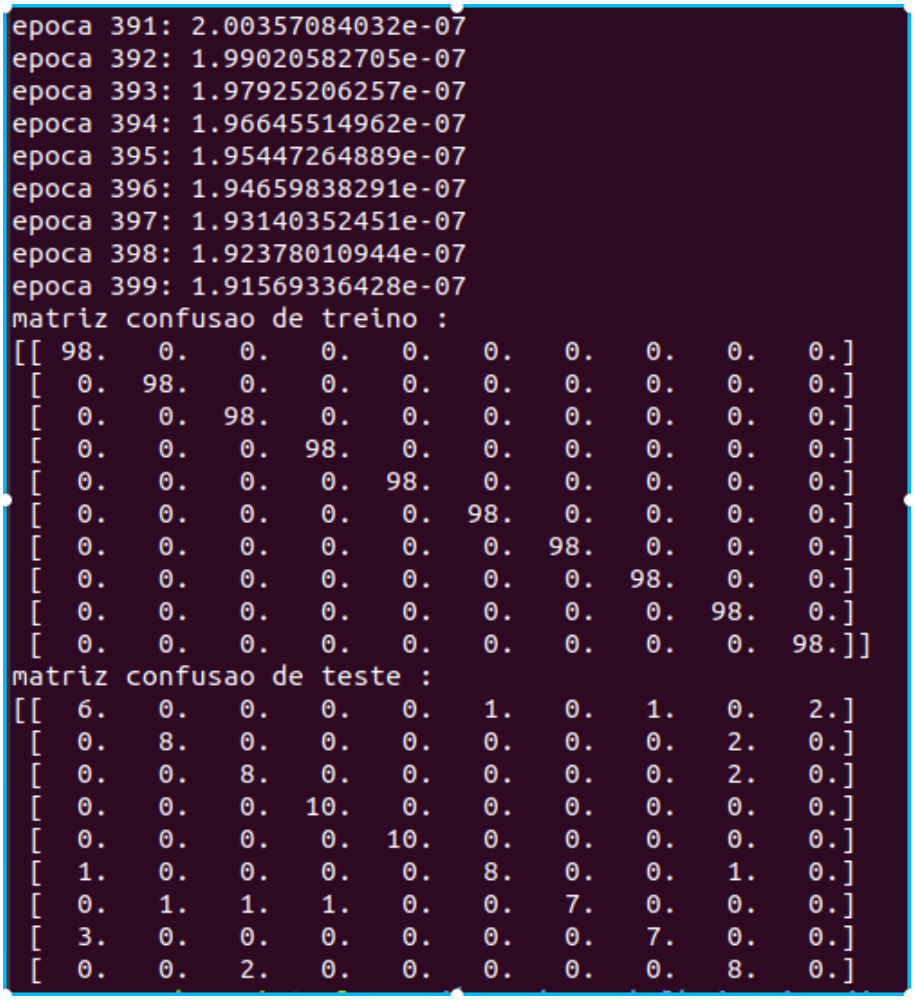


Figura 25 – Erro e matriz confusão após treinamentos da rede com overfitting.
Fonte: Própria.

Para o treinamento da rna da imagem 26, foram usadas 180 épocas e uma taxa de aprendizagem de 0,005, com uma boa taxa de acerto de 93%. O erro após a última época não está tão baixo como na rede com overfitting, da imagem 25. Ainda na imagem 26, é possível verificar que a rna não acertou todos os comandos da matriz de treino, isso significa que a rede não decorou todas as características dos padrões na etapa de treinamento, possibilitando a rede distinguir aqueles padrões que possuam um leve desvio dos comandos treinados, tornando-se importante para a rede distinguir um padrão falado por uma pessoa que possua um timbre de voz diferente daqueles timbres treinados.

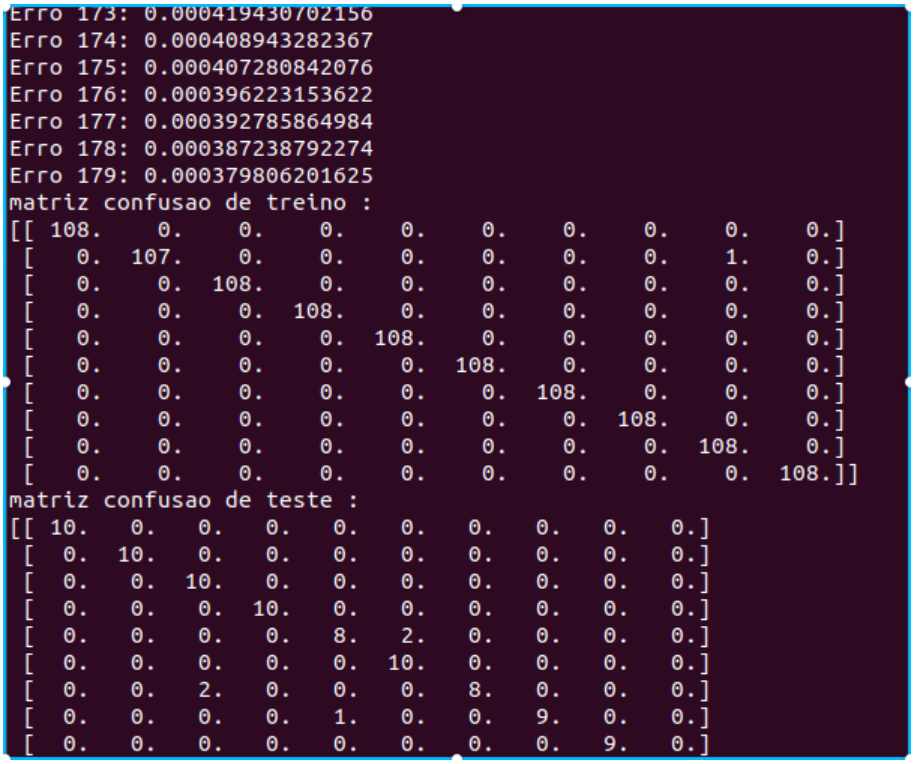


Figura 26 – Erro e matriz confusão após treinamentos da rede sem over fitting.
Fonte: Própria(2018).

A etapa de escolha da quantidade de épocas e taxa de aprendizagem, foi realizada apenas com os áudios originais, sem alteração causada pelo Sox, como discutido no tópico 3.2, Banco de Dados. Desta forma ganhou-se tempo no processo de aprendizagem, já que o tempo do processo de aprendizagem cresce junto com o banco de dados. Para um bom desempenho da RNA, o sinal de erro aceitável ficou entre 0,003 e 0,0001, com uma taxa de aprendizagem entre 0,005 e 0,008.

Com a conclusão da etapa anterior, foi escolhido o número de neurônios da camada oculta através do método da raiz quadra descrito por (MELO, 2011), ou seja, a raiz quadrada do produto dos neurônios de entrada com os neurônios de saída, obtendo um resultando de 36 neurônios para a camada oculta. Também foi testado o método da média citado por (MELO, 2011), onde o número de entrada é somado com o número de saída e divido por dois, obtendo um resultando de 70 neurônios na camada oculta, porém este método causou *overfitting* à rede, mesmo diminuindo o número de épocas e a taxa de aprendizagem.

Após todos os parâmetros definidos, o treinamento da rede neural foi realizado com os áudios manipulados pelo Sox.

4.4 DISTÂNCIA MÁXIMA E SENSIBILIDADE DO MICROFONE PARA A GRAVAÇÃO DO COMANDO

Para que o Jarbas não ficasse gravando áudio constantemente, e consumindo energia desnecessária, foi construído um algoritmo com o auxílio de biblioteca Alsa (the Advanced linux sound Architecture) Áudio, que grava e reproduz e altera arquivos Midi. A biblioteca foi instalada no python pelo comando "pip install pyalsaaudio". O algoritmo analisa a energia contida no microfone com um atraso de 140 amostras, como a frequência de amostragem para gravar o áudio é de 16000 Hz, o tempo de atraso é de 114 mili segundos, para a aplicação do Jarbas, pode-se considerar que a análise é feita em tempo real.

Para ambientes ruidosos, o limiar de energia usado para definir quando será gravado um áudio, deve ser auto o suficiente para que a gravação não seja acionada com o ruído de fundo, porém o comando só poderá ser gravado de uma distância próxima ao microfone. Nos testes realizados em ambientes extremamente ruidosos, o limiar foi definido para 0,08. Para ambientes sem muito ruído, uma residência por exemplo, o teste realizado apontou um limiar de 0,005 como adequado, dessa forma o comando poderá ser gravado de uma distância de 5 metros em volume natural.

4.5 TESTE DE ATENUAÇÃO

O banco de dados foi gravado sem a case, para ter certeza de que a case não está atenuando muito o áudio gravado, foi realizado um teste com a case e sem a case. No teste realizado, a placa fica a aproximadamente 15 centímetros da fonte de som. O som é reproduzido por uma caixa de som multimídia, reproduzindo sempre o mesmo áudio, e com o mesmo volume.

Com um algorítimo escrito em Python, foi analisado a máxima amplitude do áudio gravado com a case, e depois sem a estrutura da case. Os valores retornados pelo algorítimo foram tão distintos entre si, que o teste se tornou inconclusivo, necessitando de melhores aparelhos técnicos para medição. A figura 27, mostra uma tabela contendo os valores dos picos de energia que o algorítimo apresentou para cada gravação realizada. Possivelmente a causa da captura de valores distintos tenha sido por causa do ambiente, que não possui tratamento acústico.

Teste de atenuação
Com a case
0,0830369591913
0,0830369591713
0,199082061648
0,0930379571723
0,0830369591123
0,1930365591615
0,0991220641646

Figura 27 – Valores de pico das gravações realizadas.
Fonte: Própria(2018).

5 CONCLUSÃO

Os estudos sobre redes neurais e processamento digital de sinais para automação residencial vem aumentando exponencialmente nos últimos anos, abrindo um leque de possibilidades para a construção de um protótipo como o Jarbas, desenvolvido para ser um assistente virtual para auxiliar idosos e pessoas com deficiência nos afazeres do dia-dia.

O objetivo central deste trabalho foi o desenvolvimento de um protótipo de um sistema embarcado, capaz de interpretar comandos curtos de voz, e enviar um sinal relacionado aquele comando dito através de uma rede WiFi, utilizando o protocolo de comunicação MQTT.

Ao ligar o Jarbas, o assistente fica em modo de hibernação, até que alguém diga "Jarbas"alto o suficiente para o assistente gravar o comando, o áudio é interpretado por uma rede neural perceptron, que só acorda o assistente caso a rna entenda que o comando dito foi "jarbas". Após o assistente acordar, ele fica aguardando que alguém diga alto o suficiente para ser gravado, o comando relacionado ao eletrodoméstico e também o comando "Ligue"ou "Desligue". Caso o último comando seja interpretado como "Ligue"ou "Desligue", o assistente emite o comando interpretado via WiFi.

Para realizar o tratamento dos dados, foi realizada a segmentação do sinal de áudio, a fim de descartar as regiões de silêncio do áudio a ser usado para treinamento da rede neural. Após a etapa de eliminação de informações irrelevantes, foi realizada a etapa de extração das principais características do áudio, otimizando assim o processamento da rede neural.

Todo o processo de treinamento da rede neural foi realizado em um computador residencial, por possuir mais poder computacional do que o processador da Orange Pi. O processo final de treinamento da rede neural levou em torno de 3 horas. Com o processo de treinamento terminado, o arquivo XML com os parâmetros livres, gerado no processo de treinamento, foi exportado para o microcomputador da Orange Pi.

Nos testes realizados em ambientes limpos, e também em ambientes ruidosos, mostraram que o Jarbas obteve uma taxa de acerto superior a 80%. Após o assistente reconhecer o comando dito, o comando é emitido via wifi. Os testes de recepção de pacote mqtt, foram realizados usando o aplicativo para smartphone, "MyMqtt", que recebeu todos os comandos emitidos pelo Jarbas.

Para a fabricação em larga escala, é necessária a instalação de wifi em todos os aparelhos eletrodomésticos, assim como a adaptação para interpretação do pacote mqtt, ou a construção de um dispositivo similar a um filtro de linha para tomada, que recebe o pacote MQTT, e liga ou desliga o eletrodoméstico conectado a esse filtro. Os objetivos do projeto foram alcançados, mas a partir do aprofundamento dos estudos existe a possibilidade de melhorar o sistema, utilizando outras ferramentas para treinamento da RNA, assim como outro descritor de característica como LPC, construir um banco de dados robusto e gravar o áudio em uma frequência de amostragem maior.

REFERÊNCIAS

ALMEIDA, C. R. **Extratores de características acústicas inspirados no sistema periférico auditivo**. Dissertação (Dissertação de mestrado – Curso de Pós-graduação em eng. elétrica) — Universidade Federal de Sergipe, São Cristovão, 2014.

BISHOP, C. M. **Pattern Recognition and Machine Learning**. [S.l.]: Springer, 2006.

CARVALHO, S. N. **Estudo de um sistema de conversão texto-fala baseado em HMM**. 94 p. Dissertação (Mestrado em Engenharia Elétrica) — Universidade Estadual de Campinas, Campinas, 2013.

CRYPTOGRAPHY practical. Mel Frequency Cepstral Coefficient (MFCC) tutorial. **Site practicalcryptography.com**, 2018. Disponível em: <http://www.practicalcryptography.com/miscellaneous/machine-learning/guide-mel-frequency-cepstral-coefficients-mfccs/>.

GITHUB site. python$_speech_features$.**Site GitHub**, 2018.$Disponvelem$: <>.

GUEDES *et al.* O papel social da automação-automação inclusiva e mais sustentável. **Seminário nacional de construções sustentáveis**, Second 2012.

HAYKIN, S. **Redes Neurais - 2ed.** BOOKMAN COMPANHIA ED, 2001. ISBN 9788573077186. Disponível em: <https://books.google.com.br/books?id=lBp0X5qfyjUC,Acessadoem08deAbr2018>.

IME. **EP2: Árvores de decisão de algoritmos de ordenação**. 2018. Abril 08, 2018. Disponível em: <<https://www.ime.usp.br/~fmario/mac122-15/ep2.html>.

LIGHT., R. A. Mosquitto: server and client implementation of the mqtt protocol. **The Journal of Open Source Software,Vol2**, May 2013.

MELO, D. B. **Um Sistema de Reconhecimento de Comandos de Voz Utilizando a Rede Neural ELM.** 2011. Monografia (Engenharia de Teleinformática), FACETS (Universidade Federal do Ceará), Ceará, Brasil.

MINSKY, M.; PAPERT, S. **Perceptrons - an introduction to computational geometry**. [S.l.]: MIT Press, 1987.

MORAES, C. de; CASTRUCCI, P. de L. **Engenharia de automação industrial**. LTC, 2007. ISBN 9788521615323. Disponível em: <https://books.google.com.br/books?id=irN1PgAACAAJ>.

NOCKO, C. **Produções de Àudios : Fundamentos**. DITEC-Diretoria de Tecnologia Educacional, 2006. Disponível em: <http://www.educadores.diaadia.pr.gov.br/arquivos/File/pdf/tematicos_producoesaudio.pdf>.

OPPENHEIM, A.; SCHAFER, R. **Zeitdiskrete Signalverarbeitung**. Oldenbourg, 1999. (Grundlagen der Schaltungstechnik Series). ISBN 9783486241457. Disponível em: <https://books.google.com.br/books?id=WlEimgEACAAJ,Acessadoem02deNov2018>.

PIAZZI, P. **Aprendendo inteligência: Manual de instruções do cérebro para estudantes em geral**. Editora Aleph, 2015. ISBN 9788576572220. Disponível em: <https://books.google.com.br/books?id=78uuCgAAQBAJ>.

PRADO, A. L. d. **Estudo da aplicação de redes neuronais artificiais para aopoia a decisão na liberação do perfil lipídico e de glicemia em jejum**. Dissertação (Pós-Graduação em Bioinformática) — Universidade Federal do Paraná, Curitiba-PR, 2011.

RAUBER, T. W. **Redes Neurais Artificiais. (Apostila-MiniCurso) Encontro regional de informática sociedade brasileira de computação**. Dissertação (Pós-Graduação em Informática (PPGI)) — Universidade Federal do Espiro Santo, Espirito Santo, 2004.

ROCKENBACH, S. **Arquitetura, Automação e Sustentabilidade**. Dissertação (Pós-Graduação em Arquitetura) — Universidade Federal do Rio Grande do Sul(UFRGS), Porto Alegre, 2004.

SANT'ANA, M. V. **Reconhecimento de vocabulário utilizando redes neurais.** 2018. Monografia (Engenharia de computação), FACETS (Centro Universitário de Brasília, Sociais Aplicadas), Brasilia, Brazil.

SCHAUL, T. *et al.* PyBrain. **Journal of Machine Learning Research**, Second 2010.

SILVA, A. P. Componentes mel cepstrais. v. 0, p. 2–7, February 2015. Disponível em: <https://www.researchgate.net/publication/272351208_Componentes_Mel_Cepstrais,Acessoem:28outubrode2018.>

TDUNNING. **Hidden Markov Model**. 2018. Abril 08, 2018. Disponível em: <https://commons.wikimedia.org/wiki/File:HiddenMarkovModel.png,Acessadoem01deNove2018>.

YUAN, M. Por que o mqtt é um dos melhores protocolos de rede para a internet das coisas? **DeveloperWorks, IBM**, Outubro 2017. Disponível em: <https://www.ibm.com/developerworks/br/library/iot-mqtt-why-good-for-iot/iot-mqtt-why-good-for-iot-pdf.pdf>.

ZHENG F.; ZHANG, G. S. Z. Comparison of different implementations of mfcc. **Journal of Computer Science and Technology**, v. 16, n. 6, p. 582–589, Sept 2001.

6 ANEXOS

Este último tópico demonstra os códigos usados no projeto.

6.1 ANEXO A

O código desenvolvido em python para gravar a voz dos 62 voluntários, está disposto a baixo.

```
import numpy as np
import os

speaker = raw_input('Digite o nome do volunt rio: ')

comandos = ['JARBAS','LIGUE','DESLIGUE','MUSICA','SALA','QUARTO','COZINHA',' AR-
    CONDICIONADO','TV','CAFE']

repetitions = 3

for icomando in range(0, len(comandos)):
    for repeats in range (0, repetitions):      #grava o o comando tr s vezes
        print('')
        print('DIGA ' + str(comandos[icomando]) + '...')
        print(' ')
        gravar = 'arecord -D hw:audiocodec,0 -f dat -d 2 -c 1 -r 16000 '
        cmd = str(gravar) + str(speaker) + str('-') + str(icomando) + str('-') +
     str(repeats) + str('.wav')
        os.system(cmd)
```

6.2 ANEXO B

O código em python para realizar a segmentação do comando de áudio está anexado a baixo. O algorítimo analisa o áudio da amostra com maior energia até o início do sinal, comparando a energia dos quadros com o limiar inferir de energia.

```
# -*- coding: utf-8 -*-
import matplotlib.pyplot as plt
import matplotlib.cm as cm
import numpy as np
import scipy.io.wavfile as wav
import sys
import os
from python_speech_features import mfcc
from python_speech_features import delta
from python_speech_features import logfbank

for speaker in range(1, 1621):          #quantidade de audios a serem
    segmentados
```

```
comando = str(speaker) + '.wav'     #considerando que os audios estao
nomeados como; 1.wav, 2.wav, 3.wav ...
[fs,xi] = wav.read(comando)
xi = xi / 32768.0           #normaliza  o do vetor que receber  o  udio
estalo = 50             #a grava  o feita pela orange pi causa um estalo
no  udio  de aproximadamente 40 amostras
xi[0:estalo] = 0            #zerar as amostras do audio que contem o estalo

x = []
x = np.append(xi[0], xi[1:] - 0.97 * xi[:-1])   #filtro de pre-enfase

n = np.arange(0, len(x))

quadro = 3200
step = 800
#Calcula a energia contida no vetor x
energy = np.zeros((len(x)-quadro)/step)     #define o tamanho do vetor
energia (36 amostras)
for i in range(len(energy)):
    tmp = 0
    for j in range(i*step, i*step+quadro):  #calcula a energia dentro do
quadro definido
        tmp = tmp + x[j]*x[j]       #somat rio das amostras ao quadrado
dentro do quadro
    energy[i] = tmp             #vetor com a energia a cada quadro

iMax = np.argmax(energy)    #quadro onde foi encontrado o maior valorde
energia
vMax = energy[iMax]     #maior valor de energia, entre os quadros analisados

#calcula o limiar inferior de energia
A = 0.07
lim_inferior = A*vMax

if(iMax == 0):
    start = 0

#verificacao do limiar inferior de energia para achar o inicio da regi o
falada
energia_menor = 0
#para garantir que os vales contidos na regiao falada
#nao sejam considerados como regiao de silencio, define-se (
amostras_consecutivas = 10)
amostras_consecutivas = 10
for i in range(iMax, 0, -1):        #varre o sinal do pico de energia at  o
 inicio do sinal
    #regi o falada
-----------------------------------------------------------------------
    if (energy[i] >= lim_inferior):
        energia_menor = 0
        if(i == 1):
            start = 0   #marca o inicio da segmentacao
    #regi o de silencio
-----------------------------------------------------------------------
    else:
        if(energia_menor == 0):
            start = i   #marca o inicio da segmentacao
```

```
                start = (start*step)+quadro
            energia_menor += 1
            #quando ha 10 quadros pertecentes a regiao de silencio
            if (energia_menor == amostras_consecutivas):    #quando
                break

    stop = start + 17600        #marca o final da segmentacao
    if (stop > len(x)):
        stop = len(x)

    #vetor com a regi o de silencio zerada
    y = np.zeros(len(x))
    for i in range(start, stop):
        y[i] = x[i]

    #vetor com o audio segmentado, regiao falada
    z = np.zeros(stop-start)
    for i in range(0,len(z)):
        z[i] = y[i+start]
```

6.3 ANEXO C

Neste anexo está contido o código desenvolvido em python para a extração das características cepstrais. Este algorítimo acessa as pastas "Jarbas-10", "Ligue-11", "Desligue-12", e extrai as características de cada áudio.

```
# -*- coding: utf-8 -*-
import matplotlib.pyplot as plt
import matplotlib.cm as cm
import numpy as np
import scipy.io.wavfile as wav
import sys
import os
import math
from python_speech_features import mfcc
from python_speech_features import delta
from python_speech_features import logfbank


for comando in range(10):
    if (comando == 0):
        comando_aux = '10-Jarbas'
    if (comando == 1):
        comando_aux = '11-Ligue'
    if (comando == 2):
        comando_aux = '12-Desligue'
    if (comando == 3):
        comando_aux = '13-M sica'
    if (comando == 4):
        comando_aux = '14-Sala'
    if (comando == 5):
        comando_aux = '15-Quarto'
    if (comando == 6):
```

```
    comando_aux = '16-Cozinha'
if (comando == 7):
    comando_aux = '17-Ar-Condicionado'
if (comando == 8):
    comando_aux = '18-TV'
if (comando == 9):
    comando_aux = '19-Caf  '

for speaker in range(1, 1621):  #quantidade de arquivos em cada pasta
    #considerando que os audios estao nomeados como; 1.wav, 2.wav, 3.wav ...
    audio = str(comando_aux)+str('/')+str(speaker)+'.wav'
    [fs,xi] = wav.read(audio)
    xi = xi / 32768.0   #normaliza  o do vetor que receber  o  udio
    estalo = 50     #a grava  o feita pela orange pi causa um estalo no
 udio  de aproximadamente 40 amostras
    xi[0:estalo] = 0    #zera as primeiras 50 amostras do audio

    x = []
    x = np.append(xi[0], xi[1:] - 0.97 * xi[:-1])   #filtro de pre-enfase

    n = np.arange(0, len(x))

    quadro = 3200
    step = 800
    #Calcula a energia contida no vetor x
    energy = np.zeros((len(x)-quadro)/step)     #define o tamanho do vetor
energia (36 amostras)
    for i in range(len(energy)):
        tmp = 0
        for j in range(i*step, i*step+quadro):  #calcula a energia dentro do
 quadro definido
            tmp = tmp + x[j]*x[j]       #somat rio das amostras ao quadrado
 dentro do quadro
        energy[i] = tmp             #vetor com a energia a cada quadro

    iMax = np.argmax(energy)    #quadro onde foi encontrado o maior valorde
energia
    vMax = energy[iMax]     #maior valor de energia, entre os quadros
analisados

    #calcula o limiar inferior de energia
    A = 0.07
    lim_inferior = A*vMax

    if(iMax == 0):
        start = 0

    #verificacao do limiar inferior de energia para achar o inicio da
regi o falada
    energia_menor = 0
    #para garantir que os vales contidos na regiao falada
    #nao sejam considerados como regiao de silencio, define-se (
amostras_consecutivas = 10)
    amostras_consecutivas = 10
    for i in range(iMax, 0, -1):        #varre o sinal do pico de energia
at  o inicio do sinal
```

```
        #regi o falada
-------------------------------------------------------------------
        if (energy[i] >= lim_inferior):
            energia_menor = 0
            if(i == 1):
                start = 0   #marca o inicio da segmentacao
        #regi o de silencio
-------------------------------------------------------------------
        else:
            if(energia_menor == 0):
                start = i   #marca o inicio da segmentacao
                start = (start*step)+quadro
            energia_menor += 1
            #quando ha 10 quadros pertecentes a regiao de silencio
            if (energia_menor == amostras_consecutivas):    #quando
                break

    stop = start + 17600        #marca o final da segmentacao
    if (stop > len(x)):
        stop = len(x)

    #vetor com a regi o de silencio zerada
    y = np.zeros(len(x))
    for i in range(start, stop):
        y[i] = x[i]

    #vetor com o audio segmentado, regiao falada
    z = np.zeros(stop-start)
    for i in range(0,len(z)):
        z[i] = y[i+start]

    # normalizacao do audio segmentado entre -1 e 1
    z_norm = z/float(abs(z).max())



    # extracao das coponentes cepstrais de cada  udio
    chunks = 10 #o  udio     dividi em 10 quadros
    tempo_total = (float)(len(z_norm))/fs
    #mfcc_feat    uma matriz com as componentes cepstrais
    mfcc_feat = mfcc(z_norm,fs,winlen=tempo_total/chunks,winstep=tempo_total
/chunks,numcep=13,nfilt=26,nfft=16384,lowfreq=50,preemph=0,appendEnergy=True
,winfunc=np.hamming)

    # transformando a matriz de 13x10 amostras, que est  em duas dimens es
,
    # em um vetor de uma dimens o apenas, com 130 amostras
    for i in range(0, chunks):
        for j in range(0,13):
            sys.stdout.write(str(mfcc_feat[i][j]))
            sys.stdout.write(';')
    #inserindo o r tudo dos padroes
    if(comando == 0):
        sys.stdout.write('1;0;0;0;0;0;0;0;0;0;')
    if(comando == 1):
        sys.stdout.write('0;1;0;0;0;0;0;0;0;0;')
    if(comando == 2):
```

```
            sys.stdout.write('0;0;1;0;0;0;0;0;0;0;')
        if(comando == 3):
            sys.stdout.write('0;0;0;1;0;0;0;0;0;0;')
        if(comando == 4):
            sys.stdout.write('0;0;0;0;1;0;0;0;0;0;')
        if(comando == 5):
            sys.stdout.write('0;0;0;0;0;1;0;0;0;0;')
        if(comando == 6):
            sys.stdout.write('0;0;0;0;0;0;1;0;0;0;')
        if(comando == 7):
            sys.stdout.write('0;0;0;0;0;0;0;1;0;0;')
        if(comando == 8):
            sys.stdout.write('0;0;0;0;0;0;0;0;1;0;')
        if(comando == 9):
            sys.stdout.write('0;0;0;0;0;0;0;0;0;1;')
        print('')
```

6.4 ANEXO D

O anexo D contém o algorítmo em python usado para realizar o treinamento da rede neural.

```
from pybrain.tools.shortcuts import buildNetwork
from pybrain.datasets import SupervisedDataSet
from pybrain.supervised.trainers import BackpropTrainer
from pybrain.structure.modules import SoftmaxLayer
from pybrain.structure.modules import SigmoidLayer
from pybrain.structure.modules import TanhLayer
from pybrain.structure.modules import BiasUnit
from pybrain.tools.customxml import NetworkWriter
from pybrain.tools.customxml import NetworkReader
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

seed = 143
np.random.seed(seed)

nInputs = 130       #camadas de entrada
hidden_layers = 36  #camadas ocultas
nOutputs = 10       #camadas de saida

#importacao dos dados CSV para o aprendizado
df = pd.read_csv('treino.csv', header=None, sep=';')
X_train = df.iloc[:, 0:nInputs].values
y_train = df.iloc[:, nInputs:(nInputs+nOutputs)].values

#normalizacao do csv de treino
X_train_norm = X_train/float(abs(X_train).max())

#Construcao da rede neural
rede = buildNetwork(nInputs, hidden_layers, nOutputs, bias=True, outclass=
    SoftmaxLayer)
```

```
base = SupervisedDataSet(nInputs, nOutputs)
#insere os dados na rede neural
for i in range(len(X_train)):
    base.addSample(X_train_norm[i],y_train[i])
#treinamento da rede neural pelo metodo back propagation
treinamento = BackpropTrainer(rede, dataset = base, learningrate = 0.008,
   momentum = 0.06, batchlearning=False)
#epocas para treinamento da rede
epocas = 180

for i in range(1, epocas):
    erro = treinamento.train()
    learning_rate[i-1] = erro
    print("Erro "+str(i)+": %s" % erro)
#imprime a matriz confusao de treinamento
print('matriz confusao de treino :')
matrizConfusao = np.zeros((10,10))
for i in range(len(X_train)):
    y_certo = np.argmax(y_train[i])
    y_predito = np.argmax(rede.activate(X_train_norm[i]))
    matrizConfusao[y_certo][y_predito] += 1
print(matrizConfusao)

#importacao dos dados CSV para o aprendizado
df2 = pd.read_csv('teste.csv', header=None, sep=';')
X_train2 = df2.iloc[:, 0:nInputs].values
y_train2 = df2.iloc[:, nInputs:(nInputs+nOutputs)].values

#normalizacao do csv de teste
X_teste_norm = X_train2/float(abs(X_train2).max())

#imprime a matriz confusao de teste
print('matriz confusao de teste :')
matrizConfusao2 = np.zeros((10,10))
y_certo2 = 0
y_predito2 = 0
for i in range(len(X_train2)):
    y_certo2 = np.argmax(y_train2[i])
    y_predito2 = np.argmax(rede.activate(X_teste_norm[i]))
    matrizConfusao2[y_certo2][y_predito2] += 1
print(matrizConfusao2)

#gera um arquivo XML
NetworkWriter.writeToFile(rede, 'model.xml')
```

6.5 ANEXO E

Neste anexo está contido o algorítmo final do projeto.

```
import matplotlib.pyplot as plt
import matplotlib.cm as cm
import numpy as np
import pandas as pd
import scipy.io.wavfile as wav
```

```
import sys
import os
import math
import matplotlib.ticker as mticker
import time
import alsaaudio
from python_speech_features import mfcc, delta, logfbank
from pybrain.tools.shortcuts import buildNetwork
from pybrain.datasets import SupervisedDataSet
from pybrain.supervised.trainers import BackpropTrainer
from pybrain.structure.modules import SoftmaxLayer, SigmoidLayer
from pybrain.tools.customxml import NetworkWriter, NetworkReader
from functools import partial
from random import randrange, uniform,randint
from subprocess import call

seed = 143
np.random.seed(seed)

nInputs = 130       #camadas de entrada
hidden_layers = 36  #camadas ocultas
nOutputs = 10       #camadas de saida

#importacao do arquivo .XML, gerado pelo algor tmo de treino da rede neural
net = NetworkReader.readFrom('model.xml')

limiar = 0.01   #limiar de energia que decide quando o microfone vai gravar
periodo = 70
fs = 16000  #frequ ncia de amostragem
grava = 0
jarbas = False
flag_comando = False
cont_gravacao = 0

os.system('sudo service mosquitto restart/')            #iniciar o broker MQTT
os.system('sudo chmod 777 -R /sys/class/gpio/')         #da permissao p acessar
    a pasta gpio
os.system('echo 9 > /sys/class/gpio/export')            #exportar o gpio 9 (led
    azul)
os.system('echo 10 > /sys/class/gpio/export')           #exportar o gpio 10
os.system('echo 20 > /sys/class/gpio/export')           #exportar o gpio 20 (led
     verde)
os.system('sudo chmod 777 -R /sys/class/gpio/gpio9')        #permissao para
    acessar o gpio 9
os.system('sudo chmod 777 -R /sys/class/gpio/gpio10')       #permissao para
    acessar o gpio 10
os.system('sudo chmod 777 -R /sys/class/gpio/gpio20')       #permissao para
    acessar o gpio 20
os.system('echo out > /sys/class/gpio/gpio9/direction')     #configurar gpio 20
    com saida
os.system('echo out > /sys/class/gpio/gpio10/direction')    #configurar gpio 20
    com saida
os.system('echo out > /sys/class/gpio/gpio20/direction')    #configurar gpio 20
    com saida
os.system('echo 0 > /sys/class/gpio/gpio9/value')
os.system('echo 0 > /sys/class/gpio/gpio10/value')
os.system('echo 0 > /sys/class/gpio/gpio20/value')
```

```
while(True):
    # abre a interface de audio para leitura (1 canal, fs: 16KHz, 32 bits em
    float)
    inp = alsaaudio.PCM(alsaaudio.PCM_CAPTURE, alsaaudio.PCM_NONBLOCK)
    inp.setchannels(1)
    inp.setrate(16000)
    inp.setformat(alsaaudio.PCM_FORMAT_FLOAT_LE)
    inp.setperiodsize(periodo)

    if(jarbas == False):
        total_size = fs*1.1 #tempo de gravacao
        # abre um arquivo temporario para salvar as amostras do audio captado
        f = open('tmp.bin', 'wb')
        #acende o led azul
        os.system('echo 0 > /sys/class/gpio/gpio10/value')
        os.system('echo 0 > /sys/class/gpio/gpio20/value')
        os.system('echo 1 > /sys/class/gpio/gpio9/value')
    else:
        total_size = fs*1.1
        if(flag_comando == False):
            # abre um arquivo temporario para salvar as amostras do audio
    captado
            f = open('tmp.bin', 'wb')
            #acende o led verde
            os.system('echo 1 > /sys/class/gpio/gpio20/value')
            os.system('echo 0 > /sys/class/gpio/gpio10/value')
            os.system('echo 0 > /sys/class/gpio/gpio9/value')
        else:
            f = open('tmp2.bin', 'wb')
            # abre um arquivo temporario para salvar as amostras do audio
    captado
            #pisca o led verde
            os.system('echo 1 > /sys/class/gpio/gpio20/value')
            time.sleep(.05)
            os.system('echo 0 > /sys/class/gpio/gpio20/value')
            time.sleep(.05)
            os.system('echo 1 > /sys/class/gpio/gpio20/value')
            time.sleep(.05)
            os.system('echo 0 > /sys/class/gpio/gpio20/value')
            time.sleep(.05)
            os.system('echo 1 > /sys/class/gpio/gpio20/value')

    while total_size > 0:
        # efetua uma leitura da porta de audio
        length, data = inp.read()
        # se a leitura trouxer dados, isto eh, l > 0, salva estas amostras no
    arquivo tmp.bin ou tmp2.bim
        if(length > 0):
            # converte data para float32
            v = np.frombuffer(data, dtype=np.float32)
            # calcula a energia do quadro e, se for maior que o limiar, ativa a
    flag para gravar
            energy = np.dot(v,v) / float(len(v))
            if(energy > limiar and grava == 0):
                grava = 1
            if(grava == 1): #grava durante 1,1 segundos
```

```
            f.write(data)
            total_size = total_size - length
f.close()   # fecha o arquivo
grava = 0

if(jarbas == False or flag_comando == True):
    print('Processando audio...')
    os.system('echo 1 > /sys/class/gpio/gpio9/value')
    os.system('echo 1 > /sys/class/gpio/gpio10/value')
    os.system('echo 1 > /sys/class/gpio/gpio20/value')
    if(jarbas == False):
        lim = 1
    else:
        lim = 2
    for k in range(0, lim):
        # abre o arquivo temporario pra leitura
        if(k == 0):
            f = open('tmp.bin', 'rb')   # abre o arquivo temporario pra
leitura
        if(k == 1):
            f = open('tmp2.bin', 'rb')  # abre o arquivo temporario pra
leitura
        # carrega as amostras do audio (direto em float) para o vetor xi
        xi = np.fromfile(f, dtype=np.float32)
        f.close()   # fecha o arquivo

        x = []
        x = np.append(xi[0], xi[1:] - 0.97 * xi[:-1])   #filtro de pre-
enfase
        n = np.arange(0, len(x))
         #normaliza  o do sinal de audio
        x_norm = x/float(abs(x).max())

        chunks = 10 #n  de quadors em que o sinal de audio se a dividido
        tempo_total = (float)(len(x_norm))/fs

        #extrator MFCC - extrai as componentes cepstrais para o vetor de
duas dimens es chamado mfcc_feat
        mfcc_feat = mfcc(x_norm,fs,winlen=tempo_total/chunks,winstep=
tempo_total/chunks,numcep=13,nfilt=26,nfft=16384,lowfreq=50,highfreq=4000,
preemph=0,appendEnergy=True,winfunc=np.hamming)
        X_train = []

        #organiza mfcc_feat para que seja um vetor de um dimens o apenas
        for i in range(0, chunks):
            for j in range(0,13):
                X_train.append(mfcc_feat[i][j])
        #normalizacao das componentes cepstrias
        X_train_norm = X_train/float(abs(X_train).max())
        #insre os dados na rede
        X_train = np.reshape(X_train_norm, nInputs).astype('float32')
        activate = net.activate(X_train)    #resposta da rna

        if(jarbas == False):
            if(indice == 0):
                jarbas = True   #Jarbas acorda
            else:
```

```
                os.system('echo 0 > /sys/class/gpio/gpio9/value')
                os.system('echo 0 > /sys/class/gpio/gpio10/value')
                os.system('echo 0 > /sys/class/gpio/gpio20/value')
                #lede vermelho pisca
                os.system('echo 1 > /sys/class/gpio/gpio10/value')
                time.sleep(.03)
                os.system('echo 0 > /sys/class/gpio/gpio10/value')
                time.sleep(.03)
                os.system('echo 1 > /sys/class/gpio/gpio10/value')

        else:
            if(k == 0): #processamento do primeiro comando
                labels = ['jarbas','ligue','desligue','musica','sala','
quarto','cozinha','ar-condicionado','tv','cafe']
                comando1 = str(labels[np.argmax(activate)])
            if(k == 1): #processamento do segundo comando
                jarbas = False
                flag_comando = False
                labels = ['jarbas','ligue','desligue','musica','sala','
quarto','cozinha','ar-condicionado','tv','cafe']
                comando2 = str(labels[np.argmax(activate)])
                if(comando2 == 'ligue' or comando2 == 'desligue'):
                    mqtt = 'mosquitto_pub -h 192.168.43.142 -t jarbas -m '+
str(comando1)+'_'+str(comando2)
                    os.system(mqtt)
                else:
                    os.system('echo 0 > /sys/class/gpio/gpio9/value')
                    os.system('echo 0 > /sys/class/gpio/gpio10/value')
                    os.system('echo 0 > /sys/class/gpio/gpio20/value')
                    #lede vermelho pisca
                    os.system('echo 1 > /sys/class/gpio/gpio10/value')
                    time.sleep(.03)
                    os.system('echo 0 > /sys/class/gpio/gpio10/value')
                    time.sleep(.03)
                    os.system('echo 1 > /sys/class/gpio/gpio10/value')
else:
    flag_comando = True
```