Projeto de Graduação

DEPARTAMENTO DE ENGENHARIA ELÉTRICA

03 de Julho de 2018

PROCESSAMENTO EMBARCADO DE SINAIS CEREBRAIS POR MEIO DA UTILIZAÇÃO DE MICROCONTROLADOR

Marcos Civiletti de Carvalho

PUC RIO

Projeto de Graduação

DEPARTAMENTO
DE ENGENHARIA
ELÉTRICA

PROCESSAMENTO EMBARCADO DE SINAIS CEREBRAIS POR MEIO DA UTILIZAÇÃO DE MICROCONTROLADOR

Aluno: Marcos Civiletti de Carvalho

Orientador: Marco Antonio Meggiolaro

Trabalho apresentado como requisito parcial à conclusão do curso de Engenharia Elétrica na Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, Brasil.

Agradecimentos

Agradeço aos meus pais, a quem devo tudo que realizei durante a faculdade, pelo apoio incondicional de todas as possíveis maneiras e por tudo que não sou capaz de retribuir.

Ao meu irmão, por ser uma grande referência e me levar sempre a querer ser alguém melhor, a quem devo muito mais do que sei colocar em palavras.

Aos meus amigos, sem os quais não conseguiria trilhar essa jornada até o fim.

Ao professor Meggiolaro, por me proporcionar a oportunidade de trabalhar em um projeto de grande interesse meu e por sua orientação e paciência durante esse processo.

Resumo

Esse trabalho consiste no desenvolvimento de software em um microcontrolador que poderia ser utilizado para realizar o controle de uma interface cérebro-máquina. Foram replicados parâmetros de pré-processamento e processamento de um controle já realizado por Alexandre Ormiga Galvão Barbosa em sua dissertação de Mestrado [1], o qual havia sido desenvolvido de maneira não-embarcada, utilizando um computador. O software tem o propósito de identificar padrões de pensamento de uma mente humana através de medições de um eletroencefalógrafo (EEG).

Foi utilizada uma base de dados de sinais de EEG, referentes a atividades mentais conhecidas, com o intuito de desenvolver uma rede neural artificial para identificação desses sinais. Um bloco de leituras referente a uma atividade mental é submetido a uma série de pré-processamentos e transformado em uma entrada para a rede neural, cuja saída é uma tentativa de identificação do padrão como um dentre três possíveis.

A base de dados foi separada em um conjunto para treinamento da rede neural e outro conjunto para teste. Foi desenvolvido software em Python capaz de aplicar os pré-processamentos desejados ao conjunto de treinamento e então utilizá-los para treinamento de uma rede neural.

Por fim, implementou-se um circuito para teste da rede neural desenvolvida utilizando um segundo microcontrolador para repassar os dados do conjunto de teste à rede neural treinada carregada em um microcontrolador, simulando dados sendo recebidos diretamente de um EEG.

Palavras-chave: Microcontrolador, Sinais Cerebrais, Redes Neurais, Programação, Processamento de Sinais

On-board brain signal processing using microcontroller

Abstract

This work consists in developing software in a microcontroller which could be applied in the control of a brain-machine interface. Parameters from the pre-processing and processing were replicated from a control developed by Alexandre Ormiga Galvão Barbosa in his Master's Thesis [1], which was developed in an off-board capacity, using a computer. The software developed is intended to identify thought patterns of a human mind using electroencephalograpy (EEG) readings.

A data base of EEG signals with known outputs was used to develop an artificial neural network capable of identifying these signals. A block of samples refering to a mental activity is submitted through a series of pre-processings and adapted into one input for the neural network, which outputs an attempt to indetify the pattern as one of three possibilities.

The data base was separated in a training set for the neural network and another base for test. Software was developed in Python capable of applying the desired pre-processings to the training set and then apply them in training the neural network.

Finally, a circuit was implemented to test the developed neural network using a second microcontroller to supply the trained neural network loaded in a microcontroller with train set data, simulating reading an EEG.

Keywords: Microcontroller, Brain Signals, Neural Networks, Programming, Signal Processing

Sumário

Lista de Figuras

Lista de Tabelas

1 Introdução

a Contextualização

Devido ao compartilhamento de informações cada vez maior proporcionado pela Internet, pessoas buscam solucionar seus problemas através da implementação de projetos DIY ("do it yourself", em tradução livre “faça você mesmo”), que consistem na utilização de objetos, ferramentas e conhecimentos de fácil acesso para os mais diversos fins, como por exemplo automatização e implementação de controle inteligente no lar, sem necessidade de treino profissional ou assistência técnica.

Esses projetos DIY costumam ser caracterizados pela disponibilização de software e hardware *opensource* (do inglês, "código livre"), o que significa que qualquer pessoa pode ter acesso e sugerir melhorias no projeto. Essas melhorias são avaliadas por usuários que tenham interesse e conhecimento técnico e, caso verifique-se que as alterações são vantajosas, são implementadas, garantindo uma constante evolução.

Também é muito comum nesses projetos o uso de microcontroladores, que são, como o nome indica, dispositivos capazes de controlar (de maneira programável e inteligente) diversos outros componentes, com tamanho significativamente menor do que o de um computador.

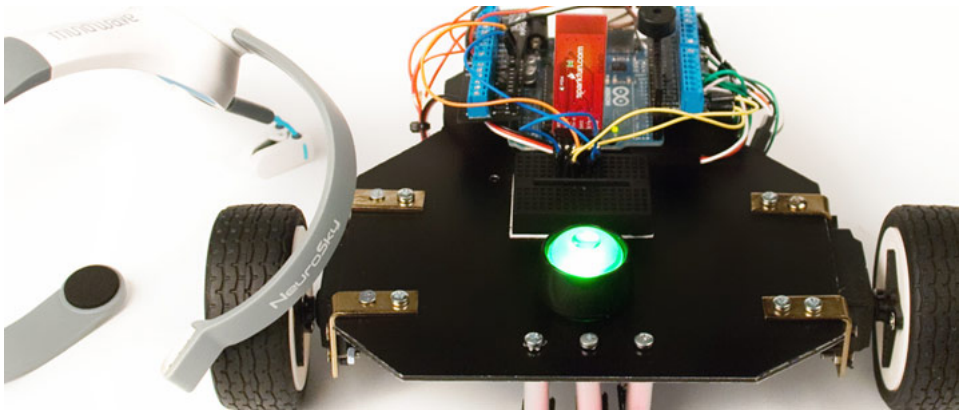

Figura 1: Projeto DIY de um robô controlado por pensamentos utilizando Arduino [2]

Um microcontrolador, em geral, é composto por um processador, memória e pinos de entrada e saída. O processador possibilita a realização dos cálculos e decisões necessários para o controle inteligente, a memória armazena instruções e padrões que o microcontrolador segue em sua programação e os pinos de entrada e saída permitem ao microcontrolador receber e enviar sinais elétricos.

O uso de sensores costuma acompanhar o uso de microcontroladores. Sensores são dispositivos transdutores que transformam grandezas físicas (como temperatura, umidade e luminosidade) em sinais elétricos. Utilizando esses sinais como entradas no microcontrolador, isso permite um controle que leve em conta essas grandezas. Também costuma utilizar-se atuadores em conjunto com os microcontroladores. Atuadores tem o propósito oposto ao de sensores, transformando sinais elétricos em grandezas físicas. Utilizando as saídas do microcontrolador como sinais de controle de atuadores, isso permite que o microcontrolador altere as grandezas físicas que o usuário desejar.

Desta forma, é possível um controle que recebe informações do ambiente e reage a elas. Um exemplo desse tipo de controle são automóveis que utilizam sensores de distância no para-choque para detectar colisões iminentes e informam ao usuário através de um aviso sonoro.

b Motivação

Por essa capacidade de processamento, memória e portas de entrada e saída, os microcontroladores podem ser compreendidos como pequenos computadores. Também em função de sua portabilidade, preço acessível e crescente capacidade de processamento, estão sendo muito utilizados para realizar atividades previamente atribuídas a computadores, sendo possível tarefas como, por exemplo, processamento de imagens para controle de um RPA (Aeronave Remotamente Pilotada, popularmente conhecida como drone) em tempo real, com o

processamento sendo realizado no próprio RPA por microcontroladores. Há registro, também, do uso de microcontroladores no processamento de sinais cerebrais, como o uso da controladora Arduino no desenvolvimento de um robô controlado por pensamentos (figura 1).

Foi feito um estudo da tese de mestrado de Alexandre Ormiga Galvão Barbosa, "Controle de um manipulador robótico através de uma interface cérebro máquina não-invasiva com aprendizagem mútua" [1], tendo em mente essas aplicações de microcontroladores. Em sua tese, Alexandre desenvolveu uma interface cérebro-máquina (usualmente denominadas BMI ou "Brain-Machine Interface"') capaz de controlar um manipulador robótico utilizando pensamentos de um usuário. Essa interface requer um processamento de sinais cerebrais, utilizando redes neurais artificiais para identificação do comando sendo enviado pelo usuário.

Pensando nas aplicações de BMIs, pode-se pensar no aprimoramento e intervenção no sistema nervoso humano. Alexandre menciona, por exemplo, a aplicação para pacientes diagnosticados com Esclerose Amiotrófica Lateral, doença que reduz significativamente a capacidade motora mas não afeta as funções cerebrais [1]. Utilizando-se uma BMI para controle de manipuladores e transportes robóticos, é possível então buscar substituir as funções motoras por atuações robóticas.

Porém, o processamento da BMI desenvolvida por Alexandre é realizado utilizando um computador. Isso significa que o usuário precisa estar conectado fisicamente a um computador para que possa se integrar à BMI, o que reduz a portabilidade do projeto e, portanto, suas aplicações. Desta forma, pensou-se no uso de microcontroladores para realizar o processamento necessário para uma BMI, de forma que isso reduziria seu custo e permitira repensar novas aplicações.

c Objetivos

Por esses motivos, buscou-se nesse trabalho realizar o processamento de sinais cerebrais utilizando um microcontrolador Raspberry Pi, modelo 3 B, implementando uma rede neural artificial. Para o desenvolvimento dessa rede, foram utilizados como referência os parâmetros do processamento realizado por Alexandre [1], buscando repensar a aplicação de sua BMI.

Foi feito o uso de um conjunto de dados previamente coletado, que será descrito no capítulo 3. Esse conjunto foi tratado, organizado e utilizado como base para o aprendizado da inteligência artificial implementada. Então, buscou-se adaptar essa rede neural de forma a possibilitar a identificação dos dados tratados e, por fim, desenvolver um circuito para teste dessa rede utilizando dados provenientes de uma fonte externa ao microcontrolador, de forma a simular a leitura de um eletroencefalógrafo.

2 Revisão Bibliográfica

a Sinais

Sinais são descrições de uma informação através de algum tipo de variação. No contexto eletrônico, entende-se por sinal a variação de uma grandeza elétrica como diferença de potencial ou corrente. São representados matematicamente por funções de variáveis independentes e costumam ser trabalhados nos domínios contínuo e discreto, devido à natureza de suas informações. [3]

Sinais contínuos são aqueles que são analisados no domínio de tempo contínuo. Isso significa dizer que entre duas medições quaisquer do sinal há infinitos pontos e que a variável independente da função assume um conjunto contínuo de valores. Em contrapartida, sinais discretos são aqueles que são analisados no domínio de tempo discreto. Ou seja, a variável independente da função só pode assumir um conjunto discreto de valores.

Microcontroladores trabalham, por natureza, de forma discreta, já que a amostragem de um sinal é feita em leituras discretas. Desta forma, mesmo um sinal contínuo precisa ser aproximado por um sinal discreto para que possa ser processado por um microcontrolador.

1 Decomposição em Frequência

Quando se trabalha com sinais elétricos, por vezes é interessante a análise do comportamento do sinal em diversas frequências. Isso significa representar o sinal como a soma de sinais periódicos harmonicamente relacionados, que são oscilações que se repetem periódicamente com um período comum entre si. A figura 2 mostra a decomposição de um sinal como soma de sinais periódicos harmonicamente relacionados.

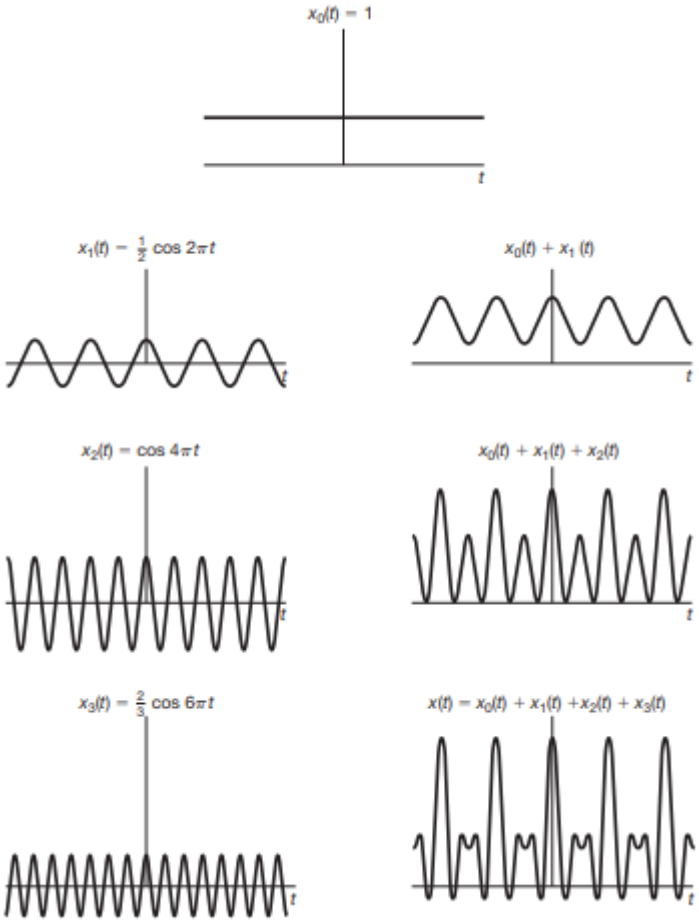

Figura 2: Representação de um sinal *x(t)* como combinação linear de sinais senoidais harmonicamente relacionados [3]

Quando se trata de um sinal gerado pela soma de outros sinais, essa decomposição por vezes permite a análise dos sinais originais, que será uma ferramenta utilizada neste trabalho ao se tratar dos sinais cerebrais.

2 Filtros

A filtragem de sinais é um processo de alterar as amplitudes relativas dos componentes em frequência de um sinal, podendo inclusive resultar na eliminação de alguns deles. Muitas vezes fatores físicos de montagem de um circuito podem resultar na geração de ruído em um sinal, que é a soma de componentes em frequência (normalmente indesejáveis) a ele. O filtro é uma ferramenta que permite a remoção dessas componentes ou a redução de suas amplitudes, permitindo a análise de um sinal mais próximo do ideal.

Três tipos de filtro relevantes para a compreensão do trabalho realizado são os filtros passa-baixas, passa-faixas e passa-altas. A nomenclatura desses filtros se refere às componentes de frequência que os filtros não amenizam ou eliminam. Filtros passa-baixa e passa-alta trabalham com a idéia de frequência de corte, que é um parâmetro do filtro que varia conforme a sua implementação. O filtro passa-baixas elimina componentes acima de sua frequência de corte, o filtro passa-altas elimina componentes abaixo de sua frequência de corte e o filtro passa-faixas implementa ambos os tipos de filtragem, eliminando componentes cuja frequência esteja fora de uma faixa determinada por ele.

b Sinais Cerebrais

Quando se trata de sinais cerebrais, é necessário compreender o que esse sinal de fato representa e como ele é adquirido.O cérebro humano funciona por meio da recepção de estímulos, processamento desses e reações. Este sistema nervoso pode ser compreendido como um sistema básico de controles, onde os nervos receptores atuam como sensores, obtendo estímulos externos e enviando esses estímulos ao cérebro humano (processador), que trabalha esses dados obtidos e então emite uma resposta (sinal de saída para os atuadores) [4].

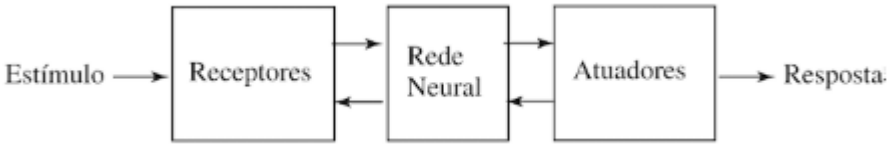

Figura 3: Diagrama de blocos mostrando funcionamento do sistema nervoso [4]

Esses estímulos são transmitidos na forma de impulsos elétricos pelos nervos até chegarem ao cérebro, onde uma rede composta de bilhões de neurônios [4] reage a esses impulsos. Essa atividade cerebral gera diferenças de potencial elétrico em diversos lugares do escalpo humano, as quais podem ser medidos utilizando sistemas de Eletroencefalografia (EEG), que consistem em medições não-invasivas desses potenciais.

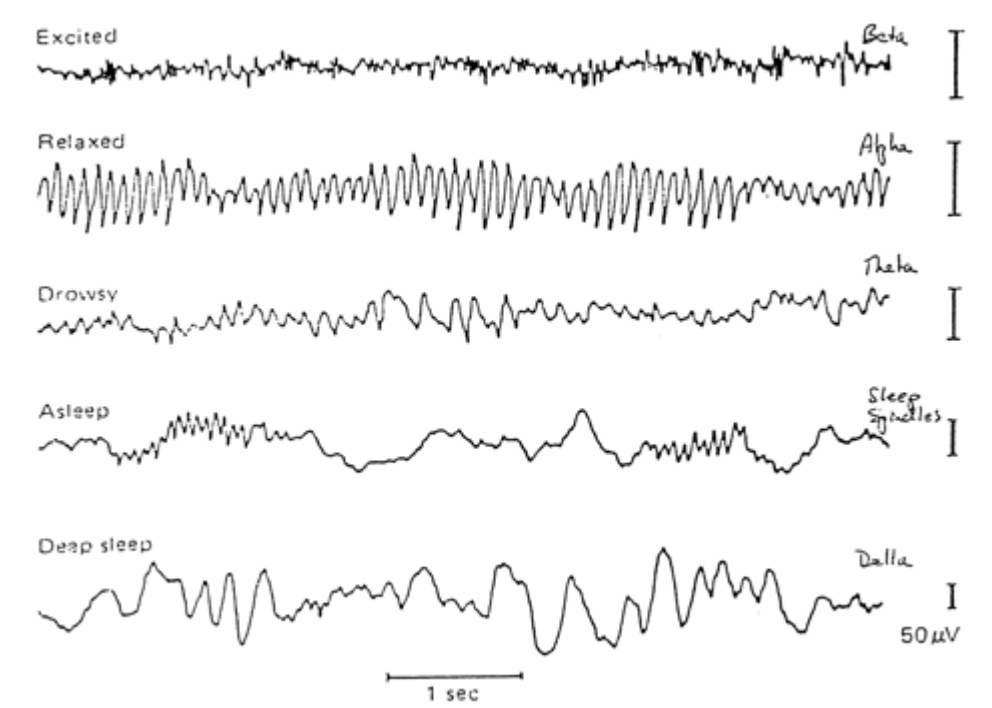

Figura 4: Leituras de EEG para diferentes estados de pensamento [5]

1 EEG

Sistemas de EEG utilizam diversos eletrodos posicionados de acordo com uma convenção, denominada *10-20 Electrode Placing System*, que identifica cada posição de eletrodo com um nome. O nome da convenção se refere ao fato de que as distâncias entre os eletrodos são sempre dimensões de 10% ou 20% de dimensões do escalpo. A figura 5 mostra a visão superior do posicionamento de eletrodos seguindo essa convenção, com a identificação das posições por seus nomes.

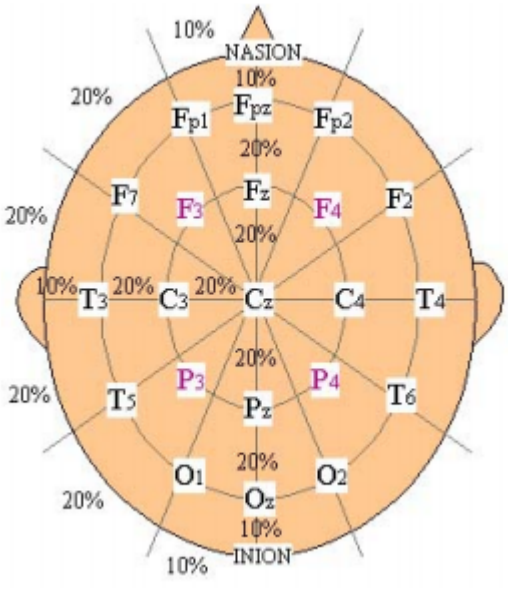

Figura 5: Posicionamento de eletrodos conforme padrão 10-20 Electrode Placing System [6]

O uso desses eletrodos permite, então, a análise de comportamento de diversas regiões do cérebro. Pode-se op-

tar por monitorar uma parte específica do cérebro realizando a análise de um conjunto selecionado de eletrodos. Dessa forma, a análise de pensamentos referentes à movimento, por exemplo, pode ser feita utilizando leituras de eletrodos posicionados em áreas do cérebro que atuam mais em pensamentos de movimento (denominadas córtex motor e pré-motor) [1].

2 Bandas de frequência do sinal EEG

Sinais EEG são complexos por natureza, devido ao fato de haver diversas interações entre neurônios ocorrendo simultaneamente. Por esse motivo, utilizando-se do conceito de decomposição em frequência previamente explicado, costuma-se tratar de componentes do sinal EEG em bandas específicas de frequência, usualmente categorizdas conforme os aspectos listados a seguir [6]:

- Alpha: banda de frequência que vai de 8 Hz a 13 Hz, tipicamente com amplitude em torno de 50 µV. Atividade na banda Alpha costuma ser induzida por fechamento dos olhos e relaxamento, sendo usualmente suprimida por abertura dos olhos ou realização de atividades pensantes;
- Beta: banda de frequência de sinais acima de 13 Hz, com amplitude em torno de 5 a 30 µV. Atividade na banda Beta costuma ser relacionada com pensamento ativo, atenção e raciocínio lógico;
- Delta: banda de frequência de 0.5 Hz a 4 Hz, tipicamente observada em pessoas jovens, lesões cerebrais e durante sono profundo;
- Theta: banda de frequência de 4 Hz a 8 Hz, tipicamente com amplitude em torno de 20 µV. Costuma ser associada à estresse emocional, meditação e estado criativo;
- RSM: chamados de ritmos sensório-motor, se encontram na mesma banda de frequência que a faixa Alpha (de 8 Hz a 12 Hz), porém adquiridos em uma região cerebral diferente (posicionamento diferente dos eletrodos). Sua atividade é associada no planejamento do movimento, sem necessidade que esse seja executado. [1]

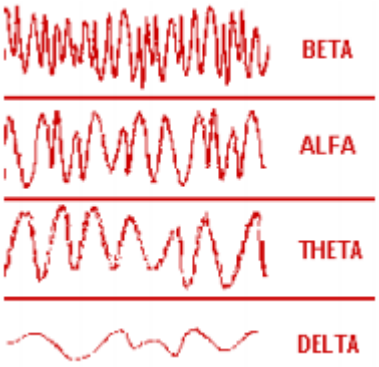

Figura 6: Exemplo de comportamento de sinais EEG nas diferentes bandas de frequência [6]

Realizando a decomposição de um sinal EEG nas bandas acima mencionadas, é possível extrair informações específicas de certos tipos de comportamento relacionados a certas bandas e relevar informações referentes à outras atividades mentais que estejam ocorrendo simultaneamente.

Como mencionado anteriormente, a escolha do posicionamento relativo do eletrodo no escalpo auxilia a extração de informações dos sinais EEG. Portanto, para analisar um tipo de comportamento do cérebro deve ser feita a seleção dos eletrodos posicionados nas áreas do cérebro que mais atuam durante esse comportamento e também deve ser feita a seleção das bandas de frequência que são tipicamente mais ativas durante este comportamento, de forma a eliminar o máximo de informações irrelevantes possível.

c Redes Neurais Artificiais

Baseado no comportamento do cérebro, onde estímulos são transmitidos a uma rede de neurônios que reagem e os propagam entre si, foi desenvolvido o conceito de uma rede neural artificial, que consiste em um sistema de processamento de dados baseado na comunicação entre unidades denominadas neurônios artificiais [4].

O neurônio artificial, cujo funcionamento é baseado no neurônio biológico humano, consiste em um pequeno processador capaz de receber diversos sinais de entrada, multiplicando cada sinal por um peso diferente (chamados pesos sinápticos). Os sinais são então somados e submetidos a uma função de ativação que gera, como saída, um novo sinal, que pode ser repassado como entrada de outro neurônio artificial, simulando o comportamento de um neurônio biológico.

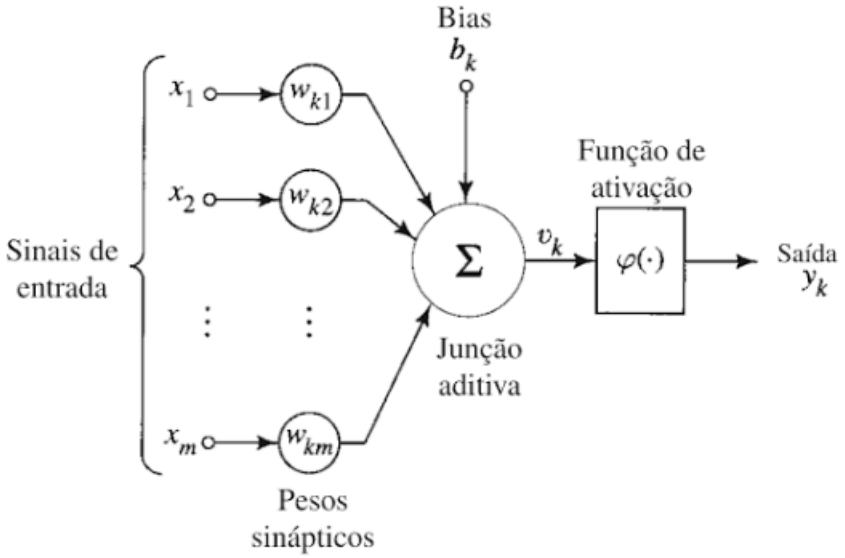

Figura 7: Modelo de um neurônio artificial [4]

O somatório realizado pelo neurônio pode incluir um termo de *bias*, que serve para aumentar ou diminuir o valor de entrada da função de ativação do neurônio. [4]

Utilizando conjuntos de neurônios artificiais, cada um com diferentes pesos sinápticos (se necessário) para os mesmos sinais de entrada, cria-se uma rede de neurônios artificiais, capaz de interpretar informações e reagir da maneira desejada, tal qual um cérebro humano. São utilizadas pois, sendo devidamente treinadas, as redes neurais têm a capacidade de generalização, ou seja, inferir informações a respeito de dados desconhecidos levando em conta um aprendizado realizado utilizando dados de mesma natureza.

Um exemplo disso é uma rede neural desenvolvida pela empresa Google com o intuito de identificar desenhos. Dados são coletados através de uma espécie de jogo, onde um usuário recebe um comando de desenhar um objeto ou animal e a rede tenta identificar o desenho, utilizando-se de um banco de dados cada vez crescente de usuários que em algum momento anterior também receberam o mesmo comando [10]. É algo semelhante a como o cérebro humano funciona, pois é capaz de identificar uma imagem que nunca viu antes levando em consideração suas experiências anteriores - como identificar um animal que já tenha visto em um desenho que nunca viu.

Um dos exemplos mais famosos no desenvolvimento de modelos de previsão é o da classificação da espécie de flores do gênero Íris, utilizando um banco de dados de 150 amostras, onde cada amostra contém as medições de comprimento e lárgura de sépala e pétala de uma flor diferente e sua espécie (*Versicolor*, *Virginica* ou *Setosa*). [11]. Utilizando uma rede neural para se adaptar a esse modelo, pode-se tentar identificar novas flores Íris medindo o comprimento e largura de suas sépalas e pétalas e passando essas informações como entradas da rede.

Figura 8: Espécies de flores do gênero Iris mostrando características da sépala e pétala [7]

Existem diversos tipos de modelos de inteligência artifical, como Lógica Fuzzy, que busca gerar um modelo de previsão que trabalhe com informações imprecisas, e Algoritmos Genéticos, que se baseia no conceito biológico de evolução para aprimoramento do seu modelo. Porém, por se tratar do processamento de sinais cerebrais, o uso de Redes Neurais faz sentido por serem um modelo que busque simular o comportamento de um cérebro humano.

A implementação de uma Rede Neural Artifical requer duas etapas, denominadas treinamento e teste.

1 Treinamento da Rede

Para que a rede desenvolvida seja capaz de agir da forma desejada, é necessário que seja treinada, utilizando-se um conjunto de dados usualmente chamado de base de treino. Essa base de treino consiste em uma série de entradas possíveis para a rede neural e suas respectivas saídas esperadas. Utiliza-se um algoritmo para que a rede ajuste os pesos sinápticos por tentativa e erro de forma a maximizar sua confiabilidade.

A estrutura básica de um algoritmo de treinamento segue três etapas:

- Fornecer os dados da base de treinamento como entrada à rede;
- Comparar as saídas geradas pela rede com as saídas esperadas para os dados;
- Alterar os pesos sinápticos dos neurônios de acordo com a discrepância verificada.

Esse algoritmo, portanto, não altera a estrutura da rede (o número de neurônios ou a maneira como esses neurônios estão interligados), mas sim a maneira como cada neurônio irá realizar o seu processamento.

Ao implementar um algoritmo de treinamento, é necessário que se determine alguns parâmetros que irão afetar a forma como a rede busca se adaptar aos dados, sendo eles:

- Épocas de Treinamento: uma execução das três etapas do algoritmo de treinamento utilizando toda a base de treinamento;
- Taxa de Aprendizado: determina o peso que o erro calculado na segunda etapa tem na alteração dos pesos sinápticos ao final de uma época do treinamento. A escolha desse parâmetro é importante para assegurar que o procedimento consiga convergir em uma configuração com mínimo de erro;
- Termo de Momentum: atua como um peso que permite que consecutivas alterações de um peso sináptico no mesmo sentido (aumentar ou diminuir consecutivamente) tenham maior contribuição na alteração dos pesos sinápticos. Há certas configurações que aparentam ser ótimas pois variações pequenas dos parâmetros apenas aumentam o erro. Porém, há situações onde, caso o aumento seja grande, encontre-se uma nova configuração ainda mais acertiva. A escolha do termo de momentum ajuda a evitar que o algoritmo se prenda nestas situações.

Os resultados da etapa de treinamento são importantes para avaliar a capacidade da rede de armazenar todos os padrões necessários para identificação dos dados. Uma rede subprojetada (com poucos neurônios ou camadas de neurônios) não será capaz de obter um bom resultado na etapa de treinamento, independentemente da escolha de parâmetros de treinamento, pois a rede não tem capacidade de levar em consideração todas as possibilidades que precisam ser avaliadas para identificação do dado.

2 Teste da Rede

Após a etapa de treinamento, a rede precisa passar por uma etapa de teste. Nessa etapa, é fornecido um outro conjunto de dados, referente ao mesmo problema que a rede busca identificar, usualmente denominado base de testes. Essa base de testes consiste também em uma série de entradas possíveis para a rede neural e suas respectivas saídas esperadas. Os dados de entrada são fornecidos à rede, que irá gerar uma saída. Essa saída é comparada com a saída esperada para avaliar se a identificação foi correta ou não. Nessa etapa nenhuma alteração é feita nos parâmetros da rede, apenas é realizada a avaliação de seu desempenho.

Os resultados da etapa de teste são importantes para avaliar a capacidade da rede de generalização - ou seja, como a rede se comporta ao receber dados previamente desconhecidos. Uma rede com boa acertividade no treinamento mas desempenho insatisfatório no teste não será capaz de identificar padrões em dados diferentes daqueles que utilizou para treinar. Assim, ao submeter-se um dado cuja saída é desconhecida, a saída prevista pela rede não será confiável.

Atingindo-se um resultado insatisfatório na etapa de teste, é então necessário retomar a etapa de treinamento da rede, alterando os parâmetros de treinamento ou até mesmo os parâmetros estruturais da rede, dependendo dos resultados apresentados.

3 Base de Dados

Para realização do treino e teste da rede neural desenvolvida, é necessário um banco de dados contendo leituras de sinais EEG referentes a certos tipos específicos de pensamentos. Para tal fim foi utilizado um conjunto de dados fornecido pelo grupo de pesquisa Berlin Brain Computer Interface (BBCI), inicialmente gerado para uso na competição BCI Competition IV [12].

A BCI Competition é uma competição de desenvolvimento de interfaces cérebro-máquina utilizando bases de dados semelhantes para preparo e validação das interfaces desenvolvidas. Para uso neste projeto foi escolhido o data set 1, cujo propósito era avaliar a classificação de sinais contínuos de EEG.

Os dados foram coletados de 59 eletrodos, posicionados de acordo com a convenção internacional 10-20 System of Electrode Placement (Figura 5), submetidos a um filtro passa-faixa de 0.05 a 200 Hz, digitalizados a 1000 Hz com uma precisão de 16 bits (0.1 µV) e depois subamostrados a 100 Hz.

Há sete bases de dados, sendo quatro obtidas de pessoas saudáveis e três geradas artificialmente, com o intuito de avaliar um método de geração artificial de sinais EEG. Nos resultados da competição, foram informados que os conjuntos denominados *a*, *b*, *f*, e *g* se referem à indivíduos reais e os demais conjuntos foram gerados artificialmente. Os dados são fornecidos nos formatos ASCII (via arquivos de texto) e Matlab. Neste trabalho, foram utilizados os dados em formato ASCII.

São fornecidos, para cada indivíduo (tanto os reais quanto os artificialmente gerados), um conjunto de dados para calibração e um conjunto de dados para avaliação. Foram selecionadas duas entre três possíveis classes de movimento (mão esquerda, mão direita ou pés) para cada indivíduo.

Nas leituras referentes ao conjunto de dados para calibração, o indivíduo recebia sugestões visuais em uma tela de computador, na forma de uma seta apontando para a esquerda, direita ou para baixo por quatro segundos, durante os quais deveria pensar no movimento referente à classe indicada. Essas sugestões ficavam na tela por 4 segundos, intercaladas por mais 4 segundos sem sugestão.

Os dados de calibração de cada indivíduo são disponibilizados na forma de três arquivos texto. Um arquivo possui as leituras contínuas dos sinais EEG dos 59 canais, onde cada linha se refere a leitura de cada um dos sinais em um instante de tempo. Outro arquivo contempla as informações referentes aos instantes de tempo onde as sugestões visuais foram dadas e qual classe foi sugerida (-1 para a primeira classe ou 1 para a segunda classe), onde cada linha contém ambas as informações a respeito de uma sugestão. O terceiro arquivo detalha quais classes são representadas pelos valores -1 e 1 (mão esquerda, mão direita ou pés), a frequência de amostragem utilizada na coleta de dados e a identificação da ordem na qual as leituras dos eletrodos é fornecida, levando em consideração a convenção de posicionamento previamente mencionada (figura 5).

Neste projeto, foi utilizado somente o conjunto de dados referente ao indivíduo *a*, uma vez que trabalhar com os padrões de pensamento referentes ao mesmo indivíduo resultam em melhores resultados, já que os dados de múltiplos indivíduos podem se comportar de maneira diferente. Também foram utilizados apenas os dados de calibração, sem uso dos dados disponibilizados para avaliação, por estarem organizados em amostragens bem demarcadas, enquanto que os dados para avaliação contém variação dos períodos de sugestão e números de sugestões por classe.

Para o indivíduo *a*, a primeira classe (com valor de saída -1) representa movimento da mão esquerda e a segunda classe (com valor de saída 1) representa movimento do pé.

Levando em conta as informações disponibilizadas sobre os dados de calibração, foi desenvolvido um programa em Python com o propósito de extrair as informações desejadas dessa base de dadods. O programa leva em conta os três arquivos fornecidos para gerar um único arquivo de texto, onde cada linha possui a leitura de 8 dos 59 sinais EEG em um instante de tempo e a classe de saída referente àquele instante. Os eletrodos selecionados foram F3, F4, Fz, C3, C4, Cz, P3 e P4, devido ao seu posicionamento acima do córtex motor e do córtex pré-motor [1], de forma que estão melhor posicionados para identificação de atividade mental referente à movimento.

Para os intervalos de tempo no qual o usuário não está recebendo nenhuma sugestão foi colocada como classe de saída o valor 0, representando um pensamento que não se encaixa em nenhuma das classes desejadas. Caso apenas se desconsidere essas leituras, estaria-se desenvolvendo uma BMI que sempre tenta classificar um pensamento como movimento de pé ou de mão, o que não é o desejado. É necessário que a rede seja capaz de distinguir momentos onde o usuário não está enviando comando e momentos nos quais o usuário está enviando comando.

O arquivo final possui 190594 linhas (referentes a 190594 leituras), cada qual composta por 9 colunas (referentes a 8 sinais EEG e uma classe de saída) espaçadas por tabulação.

4 Pré-Processamentos e Ajustes

Pré-processamentos são tratamentos aplicados a uma base de dados antes de repassá-la a um modelo de previsão. Diferentemente do modelo de previsão, que gera uma previsão a partir de um conjunto de dados de entrada, o pré-processamento consiste na geração de dados que contenham características específicas dos dados originais. A escolha dos eletrodos mencionada no capítulo anterior, por exemplo, é um tipo de pré-processamento, pois envolve a eliminação de características irrelevantes dos dados.

Para escolha dos pré-processamentos (a seguir detalhados) utilizou-se como base os resultados obtidos por Alexandre Ormiga Galvão Barbosa em sua tese de Mestrado "Controle de um manipulador robótico através de uma interface cérebro máquina não-invasiva com aprendizagem mútua" [1], que este trabalho busca reproduzir.

a Transformada Wavelet Discreta

Como explicado anteriormente, os sinais EEG costumam ser trabalhados em diferentes bandas de frequência. Para isso, foi necessário aplicar algum tipo de pré-processamento capaz de decompor o sinal em suas componentes de frequência.

Normalmente, para decomposição em componentes de frequência de um sinal utiliza-se a Transformada de Fourier [3]. Porém, ela não é aplicável neste caso, pois uma das características dessa transformada é a de que o sinal deixa de ser uma função matemática cuja variável é o tempo e passa a ser uma função matemática cuja variável é a frequência. Desta forma, ainda que o sinal esteja decomposto nas frequências desejadas, não é possível analisar seu comportamento ao longo do tempo. Por esse motivo, foi escolhida a aplicação da Transformada Wavelet Discreta (Discrete Wavelet Transform - DWT) [1].

A DWT é aplicada através de um algoritmo que consiste na decomposição de um sinal em duas componentes de resolução (número de amostras) inferior. O algoritmo aplica filtros digitais do tipo passa-baixas (eliminando componentes de alta frequência) e, paralelamente, passa-altas (eliminando componentes de baixa frequência), resultando em dois novos sinais. Um sinal, gerado pelo filtro passa-baixas, consiste nas componentes que se encontrem na metade inferior da banda de frequência do sinal original. O outro sinal, gerado pelo filtro passa-altas, consiste nas demais componentes, que se encontram na metade superior da banda de frequência do sinal original.

A saída dos filtros é subamostrada por um fator 2, o que significa que as componentes resultantes da DWT tem metade do número de amostras do sinal original. A DWT pode então ser novamente aplicada nessas componentes, resultando na divisão do sinal em diversas bandas de frequências. [1]

A figura 9 mostra um sinal s[n] passando pela DWT em três níveis, onde $h_0[n]$ representa a saída do filtro passa-baixas (cujas amostras são denominadas coeficientes de aproximação) e $h_1[n]$ representa a saída do filtro passa-altas (cujas amostras são denominadas coeficientes de detalhe). Em cada etapa, a saída $h_0[n]$ é aplicada como a entrada em outra DWT.

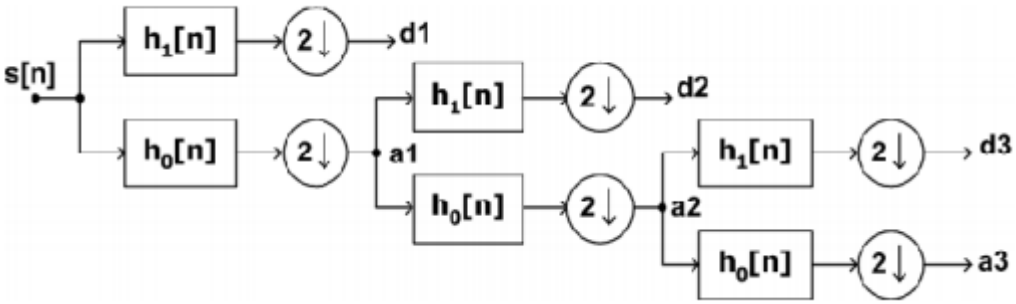

Figura 9: Diagrama de funcionamento de DWT aplicada em três níveis [1]

Utilizando essa metodologia, é possível decompor o sinal original em bandas de frequência que se aproximem das bandas de interesse dos sinais EEG. Devido aos comportamentos tipicamente associados à essas bandas, escolheu-se trabalhar com as bandas Alfa, Beta e de 0 a 32 Hz, que compreende todo o espectro EEG. [1]

Considerando o uso de uma decomposição em diversos níveis similar à apresentada acima, onde os coeficientes de aproximação são utilizados como entrada em outra DWT de forma a obter as bandas de frequência desejadas, foi pensada uma configuração que, através da aplicação da DWT em quatro níveis, permite que se encontrem as bandas desejadas. A Tabela 1 mostra um detalhamento dessa decomposição considerando um sinal inicial com n amostras. Como a base de dados utilizada foi filtrada utilizando um filtro passa-faixas de 0.05 a 200 Hz, estima-se que essa seja a banda de frequência do sinal.

Tabela 1: Decomposição do sinal em bandas de frequência com DWTs em sequência

DWT	Amostras iniciais	Amostras finais	Banda de de $h_0[n]$ (Hz)	Banda de de $h_1[n]$ (Hz)
0 (sinal original)	n	$n/2$	0 - 100	100 - 200
1	$n/2$	$n/4$	0 - 50 (EEG)	50 - 100
2	$n/4$	$n/8$	0 - 25 (EEG)	25-50
3	$n/8$	$n/16$	0 - 12.5	12.5 - 25 (Beta)
4	$n/16$	$n/32$	0 - 6.25	6.25 - 12.5 (Alfa)

Foram indicadas as bandas de frequência mais próximas às dos sinais desejados. No caso do espectro EEG, que compreende a banda de 0 a 32 Hz, foi necessário tomar uma decisão entre utilizar a banda de 0 a 25 Hz, que é mais próxima da banda ideal mas há a perda de componentes de frequência que podem ser relevantes, ou utilizar a banda de 0 a 50 Hz, que possui componentes de frequência irrelevantes. Optou-se por testar ambas as opções e averiguar qual fornece o melhor resultado.

Aplicando a transformada em mais níveis, seria possível também obter as bandas Theta e Delta. Porém, como a banda Delta costuma ser associada a sono profundo e a banda Theta costuma ser associada à estresse emocional, não são interessantes ao problema de análise de pensamento referente à movimento. Desta forma, decidiu-se por trabalhar com uma decomposição em quatro níveis, resultando em três sinais para cada um dos oito canais EEG analisados.

b Energia

Quando se trabalha com sinais EEG, normalmente utiliza-se mais de uma leitura para se referir a uma atividade mental ou a um pensamento, por não se tratar de um fenômeno instantâneo mas sim contínuo. Neste trabalho escolheu-se trabalhar com blocos de 100 leituras que, por terem sido obtidas a 100 Hz, resultam em 1 segundo de atividade mental. Esta escolha foi feita baseada no uso de 1 segundo de atividade mental para análise em [1].

Como esse segundo de amostragem representa uma atividade mental, é necessário que se extraia uma ou mais características referente a essas 100 leituras para que possa ser gerada uma entrada para a rede neural. Uma opção, por exemplo, é a média desses 100 valores, que foi um conceito avaliado por Alexandre [1] porém discartado devido à resultados insatisfatórios. Outros conceitos também avaliados foram os de Zero-Crossing e Amplitude do sinal. A decisão final foi a de utilizar o conceito de energia de um sinal, pois foi o processamento que gerou melhor resultados dentre os avaliados [1]. Por esse motivo, escolheu-se também trabalhar com a energia dos sinais.

A energia E de um sinal pode ser definida pela equação 1, onde n é o comprimento do vetor V cujas entradas são os valores medidos do sinal em diferentes instantes de tempo e V_i é o i-ésimo elemento desse vetor.

$$E = \frac{\sum_{i=1}^{n} V_i^2}{n} \tag{1}$$

O vetor V será o sinal decomposto nas devidas frequências, onde n é o número de amostras desse sinal. Para um segundo de atividade, o sinal na banda de 0 a 32 Hz terá 13 amostras (se utilizada a banda de 0 a 25 Hz) ou 26 amostras (se utilizada a banda de 0 a 50 Hz), o sinal Beta terá 7 amostras e o sinal Alpha terá 4 amostras.

Os pré-processamentos, portanto, coletam as leituras referentes a 8 eletrodos por um segundo. Esses 8 sinais são decompostos em três bandas de frequência diferentes, resultando em 24 novos sinais. É realizado o cálculo da energia desses 24 sinais o que resulta em um único vetor de 24 entradas, sendo cada entrada a energia de um dos sinais previamente mencionados. Esse vetor de 24 entradas será considerado como uma amostra para o processamento realizado pela rede neural.

c Ajuste de Dados

Como explicado anteriormente, devido ao propósito das redes neurais de serem capazes de identificar novos padrões a partir de um conhecimento previamente adquirido, normalmente utiliza-se dois conjuntos de dados. O primeiro, denominado conjunto de treinamento serve para treinamento da rede, onde se valida sua capacidade de armazenar os padrões necessários para previsões. O segundo, denominado conjunto de teste, é utilizado para validar a capacidade da rede de generalização a partir de novas entradas, que não são consideradas pelo algoritmo de treinamento.

Como será justificado no capítulo seguinte, optou-se por separar as últimas seis sugestões (três de cada classe) para teste da rede, removendo-se então as últimas 4810 entradas da base, resultando em um conjunto de 185784 amostras. Este conjunto de amostras foi pré-processado no novo programa desenvolvido, a partir do qual gerou-se a base de treinamento da rede neural desenvolvida.

Devido ao fato que cada bloco de 100 leituras deve corresponder a uma mesma classe de saída (pois irá resultar em um único vetor de entrada para a rede neural, que deve então ter uma única saída correspondente), foi necessário implementar uma lógica para que, em alguns casos, o vetor tivesse algumas (uma ou duas) leituras a mais, pois as marcações de sugestões não foram sempre exatas. Algumas sugestões estão intercaladas por intervalos de 801 ou 802 amostras, ao invés de exatamente 800, o que faz com que alguns vetores incluam algumas amostras a mais. Porém, como o cálculo de energia compensa o tamanho do vetor de entrada, essas amostras não afetam a lógica de pré-processamento a ser aplicada.

A partir das informações fornecidas juntamente com a base de dados [12], interpretou-se que a partir do tempo de sugestão indicado obtém-se 400 leituras (4 segundos) da classe de saída indicada e as demais leituras até a próxima sugestão não se referem a nenhuma das classes (ou seja, a classe 0 que representa pensamentos aguardando sugestão) - isso significa que para cada sugestão foram obtidos 8 vetores, mesmo o último conjunto de amostras contendo as leituras adicionais mencionadas anteriormente. Essa lógica apenas foi adotada para fins de geração da base de treinamento, já que em uma aplicação real a rede neural estaria constantemente recebendo dados e os processando.

d Software de Pré-Processamento

Considerando esses pré-processamentos, foi desenvolvido um outro programa em Python com o propósito de, a partir da base de dados da BBCI, gerar uma base de treinamento a ser usada pela rede neural desenvolvida. Para aplicação da DWT, foi utilizada uma biblioteca de código aberto desenvolvida especialmente para aplicação de DWT em python, denominada PyWavelets. [13]

A figura 10 mostra um exemplo do uso de Python para aplicação de DWT em um vetor qualquer, que representa um sinal. Na figura, os vetores denominados cA e cD representam os coeficientes de aproximação e de detalhe, respectivamente. É possível ver na figura que, quando se trata de um vetor com número ímpar de amostras, o processo de subamostragem resulta em metade do valor arredondada para cima (no caso, um vetor de 13 amostras resultou em coeficientes de 7 amostras).

```
>>> sinal
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13])
>>> cA, cD = pywt.dwt(sinal, 'db1')
>>> cA
array([  2.12132034,   4.94974747,   7.77817459,  10.60660172,
        13.43502884,  16.26345597,  18.38477631])
>>> cD
array([-0.70710678, -0.70710678, -0.70710678, -0.70710678, -0.70710678,
       -0.70710678,  0.        ])
```

Figura 10: Exemplo de aplicação de DWT em um vetor em Python

O programa realiza a leitura do arquivo texto que contém o banco de dados original, adicionando as 8 leituras de cada linha como entradas de 8 vetores, representando os sinais dos eletrodos em função do tempo. Esse procedimento é realizado em blocos de 100 leituras (ou mais, nos casos mencionados anteriormente que requerem algumas amostras a mais para garantir a organização dos dados) de maneira a obter-se 8 vetores de 100 entradas, representando 1 segundo de leitura de cada um dos 8 eletrodos, amostrados a uma taxa de 100 Hz.

Esses 8 vetores são submetidos a uma função de processamento que, para cada vetor, aplica a DWT em quatro níveis e extrai os coeficientes desejados. Para cada um dos 3 vetores de coeficientes que se obtém de cada sinal é chamada uma função que realiza o cálculo de energia, retornando um único valor. Os 24 valores resultantes são agregados e, juntamente com a saída esperada para esse vetor, são impressos no arquivo que irá conter a base de treinamento da rede neural.

O programa desenvolvido utilizou o arquivo fornecido pela BBCI que contém os instantes de tempo das sugestões dadas para implementar a lógica de ajuste considerando sugestões com mais de 800 amostras.

5 Desenvolvimento da Rede Neural

a Microcontrolador

O microcontrolador utilizado neste projeto foi um Raspberry Pi 3 Model B (Figura 11). Este microcontrolador é conhecido por sua capacidade de processamento e funcionalides integradas. Como é característico de um microcontrolador, possui um processador (quad-core de 1.2 GHz), memória (1 GB de memória RAM) e pinos de entrada e saída (40). Porém, o Raspberry Pi também conta com uma GPU Broadcom VideoCore IV, módulos Ethernet, Wi-Fi e Bluetooth integrados, portas USB e HDMI.

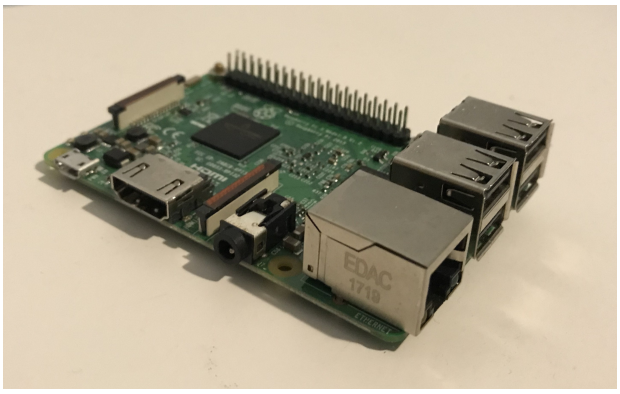

Figura 11: Raspberry Pi 3 Model B

O Raspberry Pi possui o diferencial de possuir uma configuração semelhante à de um computador, incluindo o fato de ter um sistema operacional e portas USB e HDMI que permitem o uso de monitor, teclado e mouse para uso semelhante ao de um computador, em contraposição a microcontroladores que necessitam que o usuário carregue um código que será executado toda vez que o microcontrolador for alimentado por energia.

Foi necessário o uso desse microcontrolador pois o treinamento e teste de redes neurais requer uma boa capacidade de processamento. Microcontroladores mais simples não têm uma configuração capaz de atender às demandas desse projeto, enquanto que há diversos casos de projetos DIY em que utiliza-se um Raspberry Pi para processamento de uma rede neural.

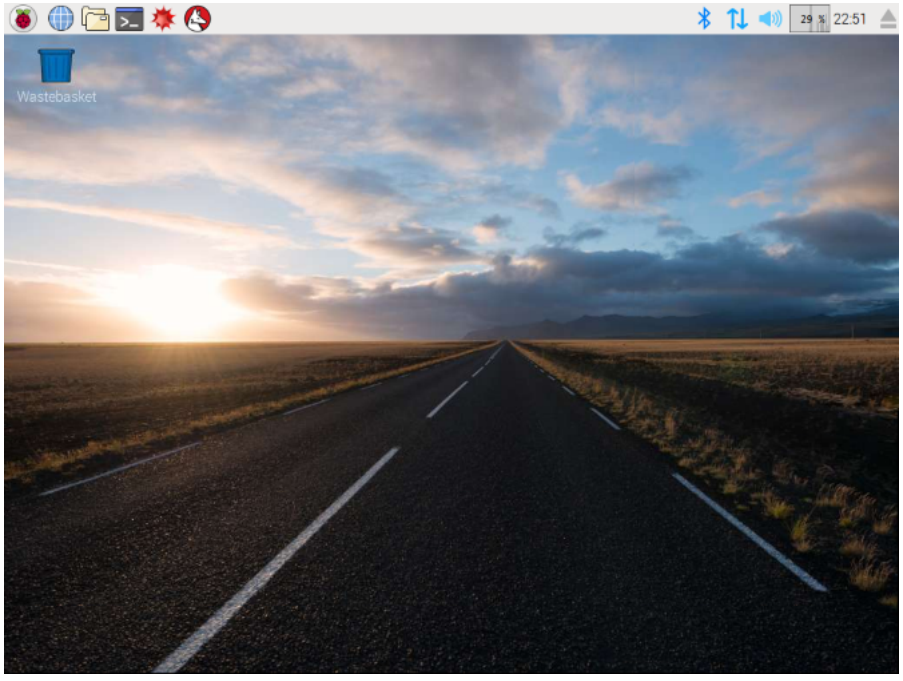

Figura 12: Desktop do sistema operacional Raspbian, similar ao de um computador [8]

Foi instalado o sistema operacional Raspbian, baseado em Linux e desenvolvido especialmente para o Raspberry Pi.

Optou-se pelo uso de um protocolo de comunicação denominado SSH que permite a um usuário conectado na mesma rede do Raspberry Pi acessá-lo e mandá-lo executar os programas desejados. Dessa forma, eliminou-se a necessidade do uso de periféricos (mouse, teclado e monitor) para testes com o Raspberry Pi.

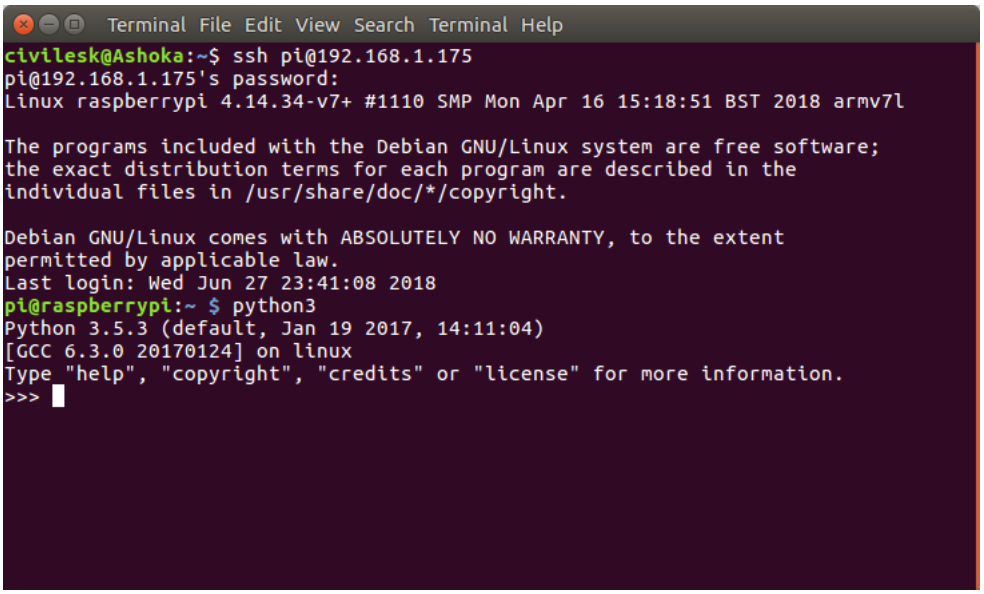

Figura 13: Terminal em Linux utilizando protocolo SSH para executar Python em um Raspberry Pi

Todos os códigos desenvolvidos foram feitos em Python 3 para que funções, bibliotecas e partes de código utilizadas nos programas desenvolvidos anteriormente pudessem ser aproveitadas no desenvolvimento do software a ser implementado no microcontrolador.

b Parâmetros da Rede Neural Artificial

Conforme previamente mencioando, este projeto se propôs a replicar um processamento já realizado em [1], e por isso a escolha de parâmetros da Rede Neural desenvolvida foi baseada nos parâmetros utilizados nesse processamento.

O tipo de rede desenvolvida foi um Perceptron de Múltiplas Camadas (*Multi Layer Perceptron* ou MLP), que é uma rede composta de múltiplas camadas de neurônios artificiais, onde os neurônios de cada camada tem como entradas todas as saídas dos neurônios da camada anterior, não estando conectados com os neurônios da mesma camada. A figura 14 mostra um tipo de MLP de 4 camadas, sendo as camadas intermediárias denominadas "camadas ocultas".

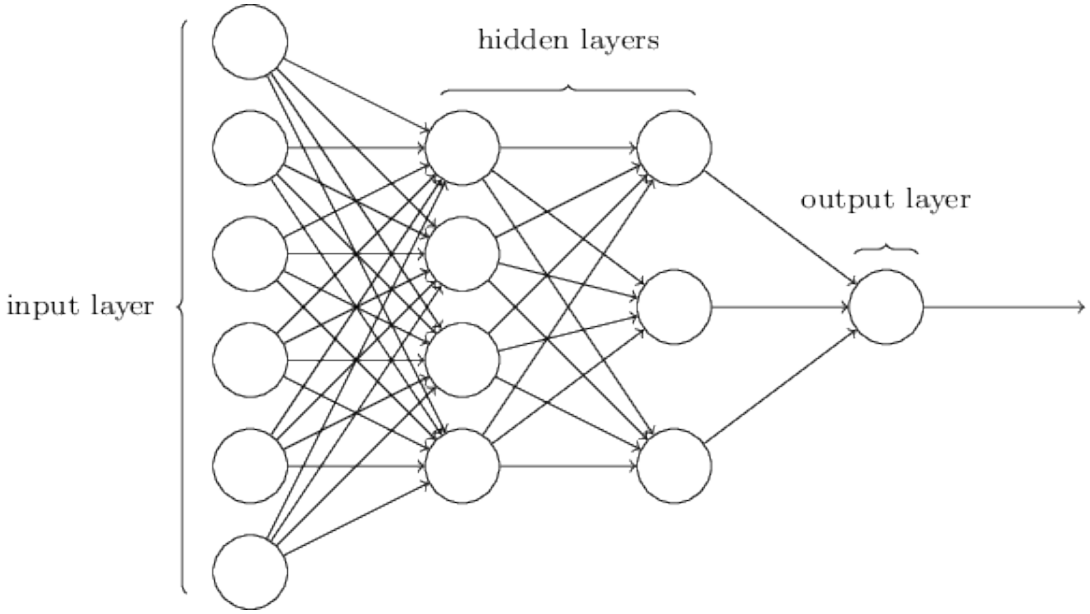

Figura 14: MLP de duas camadas ocultas [9]

A rede utilizada teve uma configuração inicial de 24 neurônios na camada de entrada, uma única camada oculta composta por 32 neurônios e 3 neurônios de saída, replicando os parâmetros do processamento de referência (porém alterando o número de neurônios na camada de saída para três, pois na base utilizada há apenas três saídas possíveis, em contraposição com o processamento de referência que tinha quatro saídas possíveis). [1]

Durante a realização dos testes, que será descrita mais a frente, variou-se o número de neurônios da camada oculta devido à resultados pouco acertivos durante o treinamento, o que pareceu indicar que a rede não tinha complexidade suficiente para armazenar os padrões necessários.

A figura 15 mostra uma representação mais próxima da configuração utilizada, contendo apenas uma camada oculta.

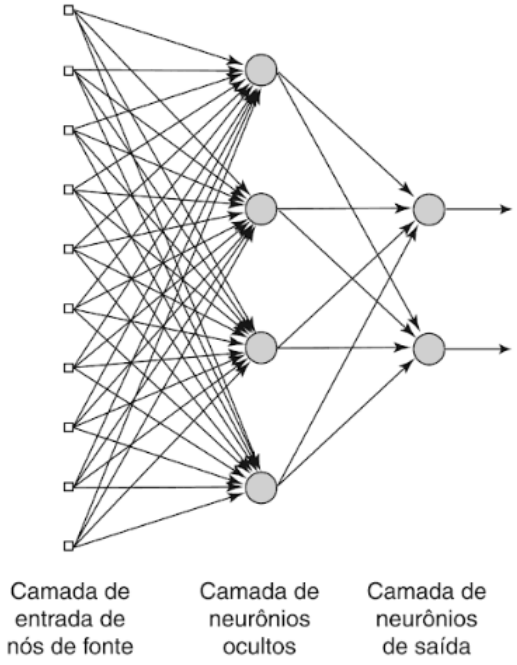

Figura 15: MLP com 10 neurônios na camada de entrada, 4 neurônios na camada oculta e 2 na camada de saída [4]

O treinamento utilizou uma taxa de aprendizado de 0.75 e termo de momentum inicial igual a 0.7, com 80 épocas de treinamento, também parâmetros baseados no processamento de [1]. Durante a etapa de testes foram realizadas alterações no termo de momentum pois verificou-se uma tendência de estagnar o treinamento em taxas de acerto baixas, problema que pode ser associado a um baixo termo de momentum.

c Programa

Para o desenvolvimento do código foi utilizada uma biblioteca de código aberto chamada Keras, que se utiliza de outras bibliotecas de aprendizado de máquina para facilitar o desenvolvimento de redes neurais. [14] Como essa biblioteca requer a instalação de uma dentre três possíveis outras bibliotecas, foi necessário também instalar a biblioteca TensorFlow [15]. Ambas as bibliotecas são compatíveis com o Raspberry Pi e, portanto, servem ao propósito do projeto.

A biblioteca Keras possui a vantagem de ser amigável ao usuário, fornecendo ferramentas intuitivas juntamente com uma capacidade de desenvolver redes de diversos níveis de complexidade, sendo muito utilizada em projetos de inteligência artificial embarcada e já possuindo de maneira integrada os algoritmos necessários para treinamento da rede e previsão de saída.

Uma outra facilidade que esta biblioteca proporciona é a capacidade de salvar e carregar modelos (tanto a arquitetura quanto os pesos sinápticos calculados após o treinamento). Isso possibilita que o Raspberry Pi não tenha que criar e treinar uma rede toda vez que for ligado, mas sim simplesmente receber uma rede que já foi previamente treinada em um computador.

Devido ao processamento exigido pelo treino de uma rede neural, costuma se recomendar que aplicações de redes neurais utilizando o Raspberry Pi incluam treinamento realizado em um computador, como foi o caso.

A estrutura do programa desenvolvido está representada pelo fluxograma da figura 16:

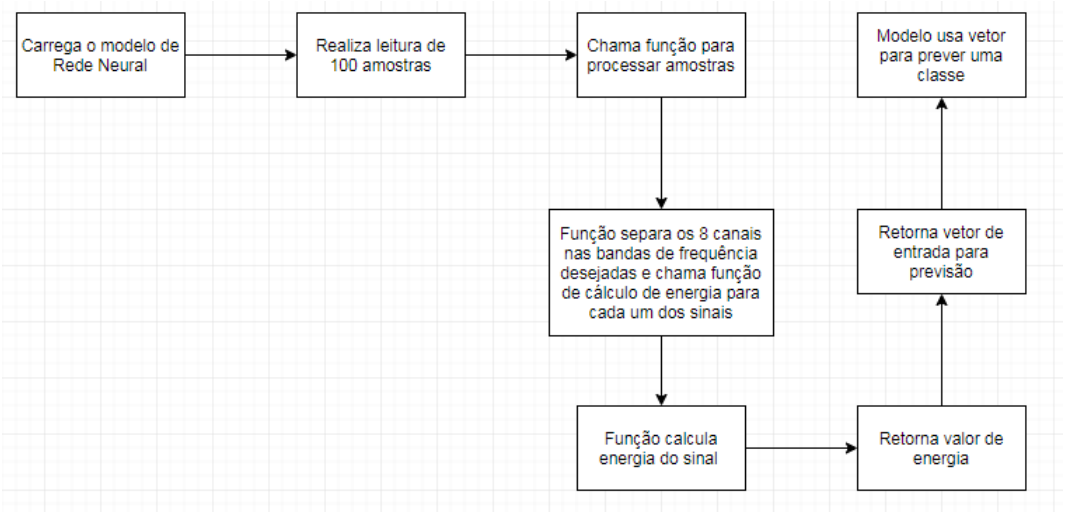

Figura 16: Fluxograma mostrando funcionamento do software desenvolvido

Utilizou-se uma ferramenta de *debugging* em python que permite, após a compilação e execução do código, que o usuário execute funções específicas manualmente, o que permitiu a realização de leituras de blocos de amostra conforme comando do usuário.

O programa desenvolvido para o Raspberry Pi consiste na definição de algumas funções, que também foram utilizadas nos programas de treino da Rede Neural e pré-processamento das bases de dados. A primeira função definida calcula a energia de um sinal, passado como argumento para a função, baseado na equação (1), dando como saída a energia do sinal de entrada.

A segunda função definida no código processa um conjunto de 100 amostras, onde cada amostra representa as leituras dos 8 eletrodos selecionados em um instante de tempo, separa-as nas bandas de interesse , chama a função que calcula a energia dos sinais e por fim retorna o vetor de 24 elementos que será uma entrada da rede neural.

O código então carrega a rede neural já treinada e então é definida uma função para a leitura de 100 amostras (que, assumindo uma frequência de amostragem de 100 Hz semelhante à da base de dados utilizada, resulta em um segundo de atividade cerebral). Essa função realiza a leitura de 100 amostras, com intervalo de leitura de 10 mS, processa esses dados utilizando as funções já definidas, insere o vetor resultante como entrada no modelo da rede neural e prevê uma saída, que é informada na tela.

Utilizando a ferramenta de debugging pode-se chamar a função de leitura a qualquer momento de um terminal, conforme desejado. Seria possível também definir uma lógica que ficasse constantemente chamando a função, simplesmente utilizando um loop condicional onde a condição é sempre verdadeira. Porém, para fins de teste (descritos a seguir), implementou-se a lógica com leitura apenas quando requisitada pelo usuário.

d Teste

1 Circuito

Para realização do teste, optou-se por uma configuração onde o Raspberry Pi recebesse os dados de uma fonte exterior, devido à uma proximidade maior dessa configuração com a realidade (em contraposição ao microcontrolador apenas ler os dados de um arquivo que já está em sua memória).

Optou-se então pelo uso de outro microcontrolador para geração dos dados a serem lidos pelo Raspberry Pi. Foi utilizado um Arduino Uno (figura 17), um dos microcontroladores mais famosos por sua facilidade de uso, baixo custo e projeto em código aberto, apesar de possuir capacidade de processamento significativamente inferior à do Raspberry Pi.

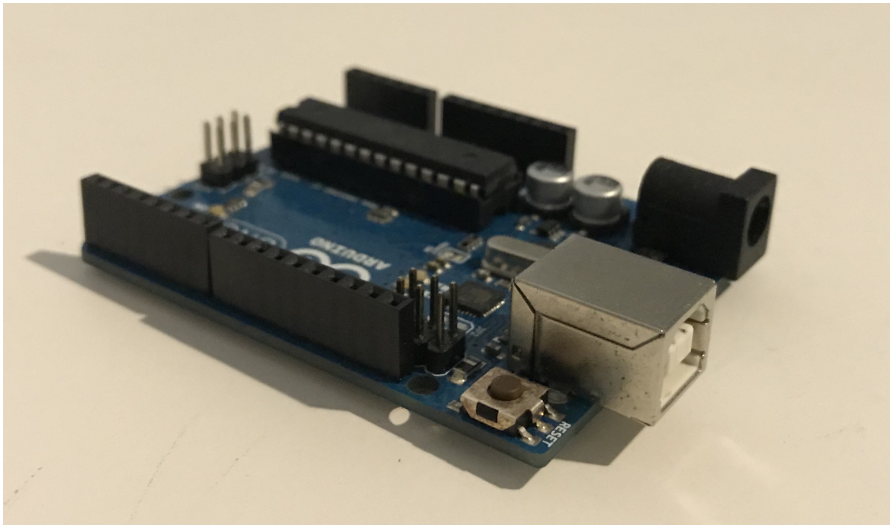

Figura 17: Arduino Uno

Como outros microcontroladores, esse modelo não possui sistema operacional, sendo necessário programar e compilar os códigos em um computador e, por meio desse, carregar os códigos no microcontrolador. A sua programação é feita em uma linguagem própria, muito semelhante a C++.

Os dados que haviam sido separados da base de treino, mencionados anteriormente (últimas 4810 entradas) foram colocados em um arquivo texto separado. A escolha de teste foi feita devido a uma simetria das últimas 6 sugestões dadas, sendo três de cada classe, facilitando a separação de dados e mantendo igual número de dados de ambas as classes. Para auxiliar a lógica de programação (que pede dados em blocos de 100 amostras) as últimas 10 leituras foram desconsideradas, resultando então em uma base de teste de 4800 amostras.

Para que o Arduino pudesse ler os dados e repassar para o Raspberry Pi, foi necessário o uso de um módulo leitor de cartão SD (figura 18), que utiliza o protocolo de comunicação SPI para possibilitar o Arduino de ler um cartão SD, permitindo então acesso ao arquivo texto de onde o Arduino extrai informações.

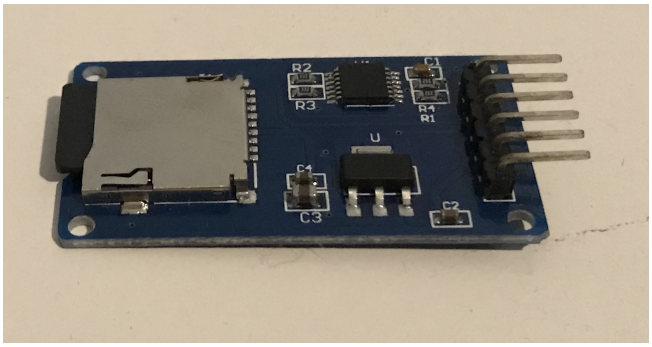

Figura 18: Módulo para leitura de cartão SD via protocolo SPI

A figura 19 mostra um fluxograma do funcionamento da comunicação implementada entre os microcontroladores:

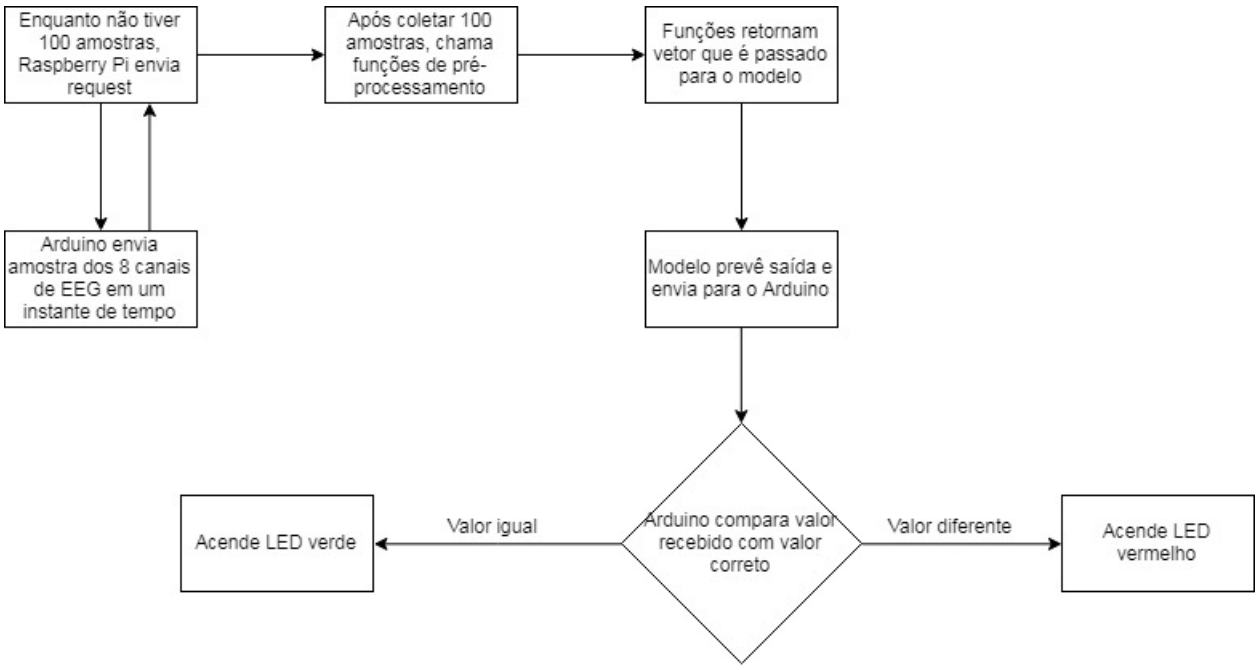

Figura 19: Representação da comunicação entre Raspberry Pi e Arduino

Foi realizada uma lógica baseada no conceito de *request* (pedido). O Raspberry Pi envia um pedido de informação para o Arduino (que seria o equivalente a realizar a leitura dos eletrodos). O Arduino então envia informação referente a uma amostra (8 números inteiros, referentes às tensões dos 8 eletrodos escolhidos) e retém a informação sobre qual classe está sendo referenciada no momento.

Esse procedimento se refere 100 vezes, após as quais o Raspberry Pi realiza uma previsão sobre o pensamento referente a essas 100 amostras e envia sua previsão para o Arduino. O Arduino compara a previsão realizada com a classe correta e, caso a previsão esteja correta, acende um LED verde. Caso esteja errada, um LED vermelho.

Para realizar essa comunicação, inicialmente se utilizou o protocolo de comunicação I^2C (*Inter-Integrated Circuit*), que consiste na definição de um dispositivo como *master* (mestre) e um ou mais dispositivos como *slaves* (escravos). Ambos os microcontroladores utilizados tem suporte à esse tipo de comunicação e bibliotecas de código para seu uso.

O *bus* de comunicação consiste na conexão dos terminais SDA e SCL em ambos os dispositivos e é controlado pelo dispositivo *master*. Normalmente é utilizado em aplicações onde um dispositivo *master* se comunica com diversos dispositivos *slaves* em um mesmo *bus*, que o *master* controla. O dispositivo *master* pode enviar a

informação diretamente para um *slave* quando quiser e o *slave* só consegue enviar informação quando recebe um *request* do *master*.

Apesar do protocolo I^2C se adequar à lógica proposta, não foi possível utilizá-lo devido à susceptibilidade do protocolo à ruídos e o volume de dados sendo transferido. Testes realizados retornavam erro de recebimento de dados sem conseguir realizar os 100 pedidos. Por esse motivo, escolheu-se o protocolo de comunicação Serial (também utilizável em ambos os microcontroladores) para novos testes.

A comunicação Serial não possui a lógica de *request* e simplesmente envia dados utilizando uma comunicação via terminais TX (transmissor) e RX (receptor). Foi implementada então, via programação, uma lógica onde o Raspberry Pi envia um comando para o Arduino (um caractere 'R') e, após receber esse comando, o Arduino envia os dados. Testes mostraram que o protocolo Serial se provou mais confiável, sendo possível realizar a leitura de todos os dados sem erro.

Foi realizada então a montagem do circuito utilizando uma placa de ensaio e cabos para conectar o Arduino com os LEDs, o módulo leitor de cartão SD e o Raspberry Pi. O Raspberry Pi foi alimentado por uma fonte externa DC 5V, o Arduino foi alimentado por uma porta USB de um computador e os demais componentes foram alimentados pelo Arduino. A figura 20 mostra uma imagem do circuito funcionando.

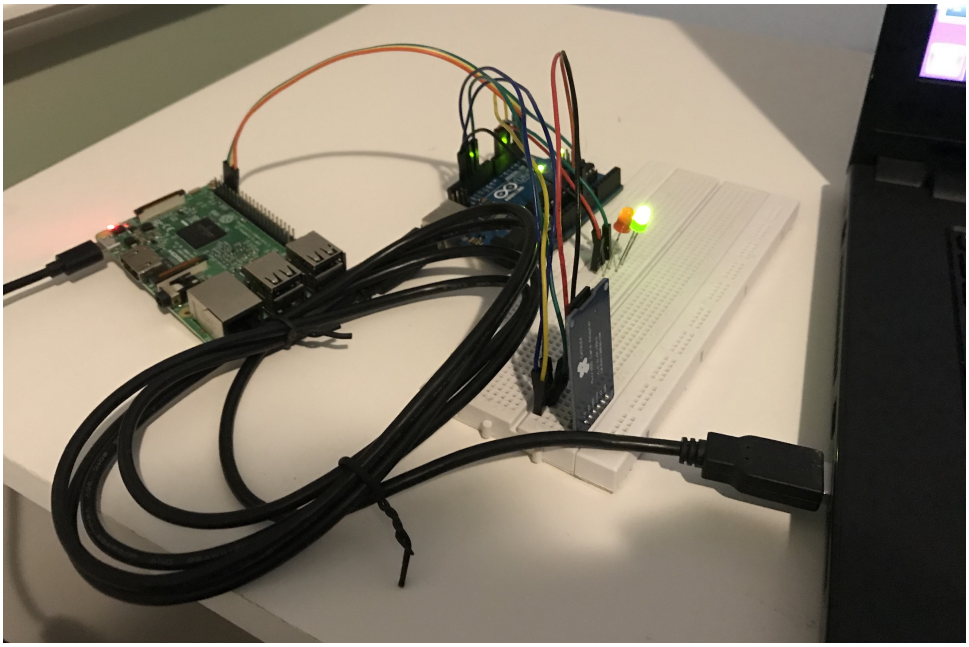

Figura 20: Circuito final funcionando

2 Metodologia de testes

Conforme mencionado, optou-se por avaliar a alteração de algumas configurações na rede neural, após testes iniciais utilizando configuração similar à de Alexandre [1]. A metodologia de testes aplicada para averiguar a configuração de melhor resultado será descrita a seguir.

Para cada configuração, foram realizadas 10 tentativas de treino e teste subsequente, sendo cada modelo aplicado ao circuito para teste, onde verificou-se a taxa de acerto entre os 48 dados de teste. Utilizou-se como critério de avaliação o resultado do modelo na etapa de teste, pois o principal objetivo de modelos de previsão é maximizar a capacidade de generalização.

Inicialmente utilizou-se a faixa de frequência de 0 - 25 Hz para representar o espectro total do sinal EEG. Utilizando um termo de momentum inicial de 0.7, optou-se por começar variando o número de neurônios na camada oculta entre 32, 48 ou 64. Então, utilizando a rede com melhor resultado, variou-se o termo de momentum entre 0.7, 0.85, 1.0 e 1.15.

Após esses testes, a lógica dos códigos foi adaptada para utilizar a faixa de frequência de 0 - 50 Hz para representar o espectro total do sinal EEG. Aplicou-se então a mesma metodologia utilizada anteriormente.

e Resultados

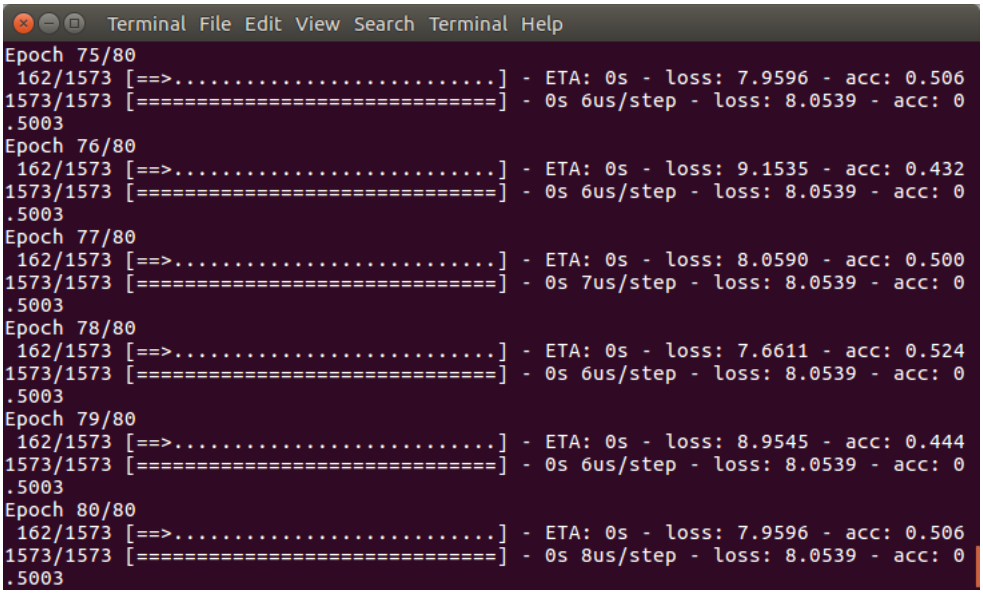

Figura 21: Informações sobre épocas de um dos treinamentos realizados

Os resultados obtidos para representação do sinal EEG na faixa de 0 a 25 Hz e termo de momentum igual a 0.7 se encontram na Tabela 2, onde N representa o número de neurônios na camada oculta de cada modelo.

Tabela 2: Resultados obtidos em primeira bateria de testes

N	Taxa de acerto no treinamento (%)	Taxa de acerto no teste (%)
32	50.29	50
48	49.78	50
64	**35.47**	**52.08**

Sendo o melhor resultado obtido com 64 neurônios na camada oculta, optou-se por essa configuração e então foi alterado o termo de momentum. Os resultados obtidos se encontram na Tabela 3, onde M representa o termo de momentum de cada modelo.

Tabela 3: Resultados obtidos em segunda bateria de testes

M	Taxa de acerto no treinamento (%)	Taxa de acerto no teste (%)
0.7	35.47	52.08
0.85	50.67	50
1.00	**47.55**	**54.17**
1.15	50.73	50

Nesta etapa foi implementada a alteração para considerar o espectro de sinais EEG na faixa de 0 a 50 Hz. Os resultados obtidos se encontram na Tabela 4, onde N representa o número de neurônios na camada oculta de cada modelo. O termo de momentum utilizado foi 0.7.

Tabela 4: Resultados obtidos em terceira bateria de testes

N	Taxa de acerto no treinamento (%)	Taxa de acerto no teste (%)
32	**50.67**	**50**
48	50.60	50
64	**50.67**	**50**

Foram obtidos resultados iguais para as configurações de 32 e 64 neurônios na camada oculta. Devido ao

melhor desempenho na primeira bateria de testes, optou-se por continuar com um modelo com 64 neurônios na camada oculta.

Por fim, foi variado o termo de momentum. Os resultados obtidos se encontram na Tabela 5, onde M representa o termo de momentum utilizada em cada modelo.

Tabela 5: Resultados obtidos na última bateria de testes

M	Taxa de acerto no treinamento (%)	Taxa de acerto no teste (%)
0.7	50.67	50
0.85	50.67	50
1.00	47.62	52.08
1.15	**50.03**	**60.42**

Conforme indicado na Tabela 5, o modelo que obteve melhor resultado foi o que utilizou 64 neurônios na camada oculta, termo de momentum igual a 1.15 e trabalhou com o sinal EEG na banda de frequência de 0 - 50 Hz. Esse modelo teve uma taxa de acerto de 60.42%, sendo capaz identificar corretamente 29 de 48 amostras de teste.

A configuração ideal obtida indica que a escolha de variação do termo de momentum e número de neurônios foi bem fundada. Apesar de que, pela própria natureza das redes neurais artificiais, seria necessário realizar uma combinação linear de diversas possíveis alterações na configuração do modelo, cada qual sendo submetida a uma bateria de testes.

Um comportamento muito comum visto nos resultados acima foi o de acerto de 50% das amostras de teste. Esse comportamento se dá em alguns modelos que, durante o treinamento, assumem uma configuração de pesos sinápticos que identifica toda amostra como classe 0. Como metade da base de testes é composta por amostras que se referem à atividades de classe 0, um modelo que sempre estima classe 0 é capaz de acertar 50% - porém, apesar de uma generalização com uma boa taxa de acerto, não é útil no desenvolvimento de BCIs porque implicaria em sempre estimar que não há um comando proveniente do usuário.

6 Conclusões

O uso de redes neurais é um trabalho extenso pois requer a configuração de uma série de parâmetros, como número de camadas ocultas, número de neurônios por camadas, taxa de aprendizado e termo de momentum de aprendizado. Existem diversas metodologias sugeridas para encontrar a configuração de rede que melhor se adequa a cada problema, mas não se trata de um problema trivial. Neste trabalho implementou-se uma metodologia para testar pequenas variações nos parâmetros de uma rede, e mesmo treinamentos diferentes utilizando modelos semelhantes obtiveram resultados diferentes.

Sugere-se o desenvolvimento de um algoritmo para teste de diferentes possíveis combinações de parâmetros da rede neural. Automatizar o procedimento realizado como metodologia de teste neste trabalho permitiria o desenvolvimento de uma rede neural com maior confiabilidade, não se adequando a um conjunto específico de dados de treino e teste apenas.

O número de redes treinadas que resultaram em modelos com previsão constante de classe 0 indicam uma necessidade de um tratamento das bases de dado utilizadas. Deve-se remover amostras com classe de saída 0 de forma a manter uma distribuição equilibrada de amostras das três classes. Outra opção que pode ser implementada, caso não se deseje perder dados, é a de usar duas redes neurais artificiais, ambas com o intuito de realizar uma classificação binária. Uma tenta identificar se um pensamento que foi recebido como entrada é um comando do usuário e, caso identifique, repassa à segunda rede neural que pode então tentar identificar a que comando se refere. Esta implementação permite, portanto, o uso completo da base de dados original, sem necessidade de remoção de amostras.

Pode ser realizado também o estudo de uso de diferentes indivíduos no treinamento desta rede neural. A princípio, mesmo o software desenvolvido tendo usado como base um indivíduo específico, ele é adaptável a outros indíviduos, bastando apenas realizar novamente o processo de treinamento da rede neural. Não se deve esperar obter os mesmos resultados, mas espera-se alguma semelhança nos comportamentos.

Uma alteração interessante seria, em uma etapa utilizando eletrodos para medição de EEG, alterar a lógica de coleta de amostras para sempre processar as últimas 100 amostras, não necessariamente dividindo-as em blocos de 100 mas sim eliminando sempre a amostra mais antiga e adicionando a mais nova ao processamento, oferecendo sugestões ao usuário transicionando de um pensamento a outro de forma mais contínua.

Sendo desenvolvido um processamento mais preciso de sinais cerebrais, pode-se pensar na realização de testes com comandos cada vez mais específicos, apesar de que o processamento de comandos gerais já permite o desenvolvimento de aplicações interessantes utilizando controle por pensamento.

Por fim, o trabalho realizado mostrou que é possível a implementação de uma rede neural artificial em um microcontrolador para identificação de padrões de pensamento. O software desenvolvido já possui uma lógica para uso de um modelo treinado e coleta de amostras para processamento nesse modelo, possibilitando que, com alguns ajustes, se utilize um EEG para leituras realizadas diretamente de um cérebro humano. Propôs-se manter o treinamento da rede neural em um computador, o que, em uma aplicação do software para controle de uma BMI, pode ser pensado como uma calibração necessária antes do uso propriamente dito. Ainda sim, a portabilidade e praticidade características de um microcontrolador se mantém relevantes após a etapa de calibração.

Referências

[1] A. O. G. Barbosa, “Controle de um manipulador robótico através de uma interface cérebro máquina não-invasiva com aprendizagem mútua,” Master’s thesis, Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, 2010.

[2] T. Karvinen and K. Karvinen, *Make a Mind-Controlled Arduino Robot*, 1st ed. O’Reilly Media, 2012.

[3] A. V. Oppenheim and A. S. Willsky, *Sinais e Sistemas*, 2nd ed. Pearson Education do Brasil, 2010.

[4] S. Haykin, *Neural Networks: a comprehensive foundation*, 2nd ed. Prentice Hall, Inc., 1999.

[5] N. Center, “Quantitative eeg.” [Online]. Available: https://neurodevelopmentcenter.com/neurofeedback-2/qeeg/

[6] M. Teplan, “Fundamentals of eeg measurement,” *Measurement Science Review*, vol. 2, no. 2, pp. 1–11, 2002.

[7] S. Fialoke, “Classification of iris varieties.” [Online]. Available: http://suruchifialoke.com/2016-10-13-machine-learning-tutorial-iris-classification/

[8] C. Cawley, “Upgrade rasperry pi’s raspbian os with the pixel desktop environment".” [Online]. Available: https://www.makeuseof.com/tag/upgrade-raspberry-pis-raspbian-os-pixel-desktop-environment

[9] R. Karim, “Deep learning via multilayer perceptron classifier.” [Online]. Available: https://dzone.com/articles/deep-learning-via-multilayer-perceptron-classifier

[10] Google, “Quick, draw!” [Online]. Available: https://quickdraw.withgoogle.com/

[11] R. A. Fisher, “The use of multiple measurements in taxonomic problems,” *Annals of Eugenics*, vol. 7, no. 7, pp. 179–188, 1936.

[12] B. Blankertz, G. Dornhege, M. Krauledat, K.-R. Müller, and G. Curio, “The non-invasive berlin brain computer interface: Fast acquisition of effective performance in untrained subjects,” *Measurement Science Review*, vol. 37, no. 2, pp. 539–550, 2007.

[13] G. Lee, F. Wasilewski, R. Gommers, K. Wohlfahrt, A. O’Leary, H. Nahrstaedt, and Contributors, “Pywavelets - wavelet transforms in python,” 2006. [Online]. Available: https://github.com/PyWavelets/pywt

[14] F. Chollet *et al.*, “Keras,” 2015. [Online]. Available: https://keras.io

[15] M. Abadi *et al.*, “TensorFlow: Large-scale machine learning on heterogeneous systems,” 2015. [Online]. Available: https://www.tensorflow.org/

A Apêndice A: Código para treino de rede neural em Python (nn_training.py)

```
def get_train_data():
        x_data = [0]*24
        y_data = []
        for x in range(24):
                x_data[x] = []
        file = open("base\_treino.txt", "r")
        for line in file:
                nums = line.split()
                y_data.append(int(nums.pop()))
                for x in range(24):
                        x_data[23-x].append(int(nums.pop()))

        return x_data, y_data

import numpy as np

from keras.models import Sequential

model = Sequential()

from keras.layers import Dense, Dropout, Activation
from keras.optimizers import SGD
from keras.utils import to_categorical

model.add(Dense(units = 24, activation='relu', input_dim=24))
model.add(Dense(units = 64, activation='relu'))
model.add(Dense(units = 3, activation='softmax'))

x_data, y_data = get_train_data()

x_data_t = list(map(list, zip(*x_data)))

x_train = np.array(x_data_t)
y_train = np.array(y_data)


y_train = to_categorical(y_train, num_classes = 3)


sgd = SGD(lr=0.75, decay=0, momentum=1.15, nesterov=True)
model.compile(loss='categorical_crossentropy',
              optimizer=sgd,
              metrics=['accuracy'])



model.fit(x_train, y_train,
          epochs=80,
          batch_size=162)
score = model.evaluate(x_train, y_train, batch_size=8, verbose=0)
print (score)

model.save('model.h5')
```

B Apêndice B: Código executado no Raspberry Pi (rpi_main.py)

```
def energia(sinal):
        n = len(sinal)
        soma = int(0)
        for x in range(n):
                entrada = int(sinal.pop())
                soma = soma + entrada*entrada
        energia = soma/n
        return energia

import pywt

def process(amostras):
        leituras = list(map(list, zip(*amostras)))
        vetor = []
        for x in range (8):
                cA1, cD1 = pywt.dwt(leituras[x],'db1')
                cA2, cD2 = pywt.dwt(cA1, 'db1')
                cA3, cD3 = pywt.dwt(cA2, 'db1')
                cA4, cD4 = pywt.dwt(cA3, 'db1')
                cA5, cD5 = pywt.dwt(cA4, 'db1')
                Beta = cD4.tolist()
                Alpha = cD5.tolist()
                                                        EEG = cA2.tolist()
                vetor.append(energia(EEG))
                vetor.append(energia(Alpha))
                vetor.append(energia(Beta))
        return vetor


import numpy as np

from keras.models import Sequential
from keras.models import load_model

model = load_model('modelo.h5')


import serial
import struct
from time import sleep

ser = serial.Serial("/dev/ttyS0", 9600)

cnt = 0.010

def get_data():
        amostras = []
        for i in range(100):
                sample = []
                ser.write("R".encode())
                data = ser.read(33)
                #print(data[0])
                for c in range(8):
                        raw = data[4*c+1:4*c+5]
                        num = struct.unpack('i',raw)[0]
                        sample.append(num)
                sleep(cnt)
                #print(sample)
```

```
        amostras.append(sample)
vetor = process(amostras)
classes = model.predict(np.array([vetor]), batch_size = None, verbose = 0, steps= None)
if(classes[0][0] == 1):
        ser.write("0a".encode())
        print("Classe 0")
elif(classes[0][1] == 1):
        ser.write("0b".encode())
        print("Classe 1")
elif(classes[0][2] == 1):
        ser.write("0c".encode())
        print("Classe -1")
```

C Apêndice C: Código em Arduino (arduino_main.ino)

```
#include <SD.h>

#include <SPI.h>


#define PIN_RIGHT 5

#define PIN_WRONG 6



int cS = 4, count = 0, output, newoutput, rqs = 0;


char flag = 0x00, input;

byte data[4];

char data_s[34];   //'S' + 4*8 BYTES + '\0'

int num[8];


File dataFile;


int read_num(){
        char c, sinal = 0x00;
        int i = 0;
        c = dataFile.read();
        if(c == 45)
                sinal = 0xFF;
        else
                i = c - 48;
        c = dataFile.read();
        while(c != 9 && c!= 13 && c!= 10){
                i = i*10 + (c - 48);
                c = dataFile.read();
        }
```

```
        if(sinal)
                i = -1*i;
        return i;
}

void read_line(){
        int i, j;
        data_s[0] = 'S';
        for(i = 0; i < 8; i++){
                num[i] = read_num();
                for(j = 0; j < 4; j++){
                        data_s[4*i+j+1] = (num[i] >> (8*j));
                }
        }
        output = newoutput;
        newoutput = read_num();
        dataFile.read();
        data_s[33] = '\0';
        flag = 0xFF;
}

void receive(){
        int previsao;
        switch(input){
                case 'a':
                        previsao = 0;
                        break;
                case 'b':
                        previsao = 1;
                        break;
                case 'c':
```

```
                        previsao = -1;
                        break;
                default:
                        previsao = 255;
        }
        if(previsao != 255){
                rqs = 0;
                if(output == previsao){
                        digitalWrite(PIN_RIGHT, HIGH);
                        digitalWrite(PIN_WRONG, LOW);
                }
                else{
                        digitalWrite(PIN_WRONG, HIGH);
                        digitalWrite(PIN_RIGHT, LOW);
                }
        }
        else{
                digitalWrite(PIN_RIGHT, LOW);
                digitalWrite(PIN_WRONG, LOW);
        }
}

void setup() {
        Serial.begin(9600);      //Inicia comunicação Serial

        pinMode(PIN_RIGHT, OUTPUT);
        pinMode(PIN_WRONG, OUTPUT);

        digitalWrite(PIN_RIGHT, LOW);
        digitalWrite(PIN_WRONG, LOW);
```

```
        if (!SD.begin(cS)){

                \\Se SD não inicializa, acende ambos os LEDs
                digitalWrite(PIN_RIGHT, HIGH);

                digitalWrite(PIN_WRONG, HIGH);


        }




        dataFile = SD.open("datatest.txt");


}


void loop() {

        int num, i;

        char command;


        if(!flag){

                read_line();

        }


        if(Serial.available()){

                command = Serial.read();

                if(command == 'R'){

                        if(flag){

                                Serial.write(data_s, 33);

                                flag = 0x00;

                        }

                }

                else if(command == 'O'){

                        while(!Serial.available());

                        input = Serial.read();


                        receive();
```

```
                    }

            }

            delay(1);

    }
```

D Apêndice D: Esquemático do circuito utilizado para teste