UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIA DA COMPUTAÇÃO – BACHARELADO

RECONHECIMENTO DE SINAIS EM LIBRAS UTILIZANDO LEAP MOTION

WANDERSON GARCIA LIMA

BLUMENAU
2015

2015/1-28

WANDERSON GARCIA LIMA

RECONHECIMENTO DE SINAIS EM LIBRAS UTILIZANDO LEAP MOTION

Trabalho de Conclusão de Curso apresentado ao curso de graduação em Ciência da Computação do Centro de Ciências Exatas e Naturais da Universidade Regional de Blumenau como requisito parcial para a obtenção do grau de Bacharel em Ciência da Computação.

Prof. Aurélio Faustino Hoppe, Mestre – Orientador

BLUMENAU
2015

2015/1-28

RECONHECIMENTO DE SINAIS EM LIBRAS UTILIZANDO LEAP MOTION

Por

WANDERSON GARCIA LIMA

Trabalho de Conclusão de Curso aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II pela banca examinadora formada por:

Presidente:	Prof. Aurélio Faustino Hoppe, Mestre – Orientador, FURB
Membro:	Prof. Dalton Solano Reis, Mestre – FURB
Membro:	Prof. Joyce Martins, Mestre – FURB

Blumenau, 09 de julho de 2015

Dedico este trabalho à minha família que me incentivou e apoiou em todos os momentos, ao meu orientador Professor Aurélio Faustino Hoppe, aos meus amigos por sempre terem ouvido minhas reclamações sem perder a paciência e finalmente todos aqueles que me ajudaram de alguma forma durante o decorrer do curso.

AGRADECIMENTOS

A minha família que sempre acreditou, me incentivou e apoiou em todos os momentos.

Agradeço ao meu orientador Aurélio por todo apoio.

Aos meus amigos que sempre me incentivaram.

Aos colegas de cursos e trabalhos que me ajudaram em diversos momentos.

Uma vez Flamengo, Flamengo até morrer.

Lamartine Babo

RESUMO

Esse trabalho apresenta o desenvolvimento de um protótipo para reconhecimento de sinais em Libras utilizando Rede Neural Artificial (RNA) com o algoritmo *backpropagation*. Desenvolvido em C++, o protótipo permite cadastrar sinais, treinar a rede neural artificial e realizar o reconhecimento de sinais oriundos do dispositivo Leap Motion. A partir dos experimentos realizados, comprovou-se que o protótipo desenvolvido é capaz de reconhecer sinais estáticos e dinâmicos.

Palavras-chave: Libras. Leap Motion. Reconhecimento de sinais. *Backpropagation*.

ABSTRACT

This work presents the development of a prototype for signs recognition in Pounds using Artificial Neural Network (ANN) with backpropagation algorithm. Developed in C ++, the prototype allows registering signals, train artificial neural network and realize the recognition signals from the Leap Motion device. From the experiments, it was shown that the developed prototype is able to recognize static and dynamic signals.

Key-words: Libras. Leap Motion. Sign recognization. Backpropagation.

LISTA DE FIGURAS

Figura 1 - Sinal referente a aprender em LIBRAS 17
Figura 2 - Sinal referente a importante em LIBRAS 17
Figura 3 - Sinal referente a comunicar em LIBRAS 17
Figura 4 - Projeção de mão capturada pelo Leap Motion 18
Figura 5 - Demonstração dos eixos x, y e z no Leap Motion 19
Figura 6 - Exportando dados do Leap Motion com LeapJS Playback 19
Figura 7 - Estrutura de sistema de reconhecimento de sinais 22
Figura 8 - Detecção do símbolo Y 25
Figura 9 - Imagem representando o símbolo V e cada passo do tratamento de imagem 27
Figura 10 - Operações morfológicas sobre amostras do sinal B 28
Figura 11 - Esqueletonização de amostras do sinal B 28
Figura 12 - Histograma de orientação sobre amostras do sinal B 29
Figura 13 - Busca pela palavra cerveja 30
Figura 14 - Ordenação pela configuração da mão 30
Figura 15 - Diagrama de caso de uso do protótipo 33
Figura 16 - Estrutura do sinal 35
Figura 17 - Estrutura de treinamento e reconhecimento de sinais 37
Figura 18 - Estrutura dos dados 38
Figura 19 - Diagrama de atividades reconhecer sinal 39
Figura 20 - Topologia RNA com 4 camadas 43
Figura 21 - Fórmula sigmóide 47
Figura 22 - Representação gráfica da função sigmóide 47
Figura 23 - Representação gráfica da curvatura do erro contendo mínimo local 48
Figura 24 - Tela inicial do protótipo 50
Figura 25 - Tela do cadastro de sinais 50
Figura 26 - Tela de cadastro de amostras 51
Figura 27 - Tela de treinamento da rede neural artificial 52
Figura 28 - Reconhecimento do sinal L 53
Figura 29 - Erro ao reconhecer sinal referente à letra T 56

LISTA DE QUADROS

Quadro 1 - Estrutura gramática em LIBRAS. Eu irei para casa 18
Quadro 2 - Estrutura gramática em LIBRAS. Quantos anos você tem? 18
Quadro 3 - Representação `metadata` e `frame` 20
Quadro 4 – Interior do `metadata` 20
Quadro 5 - Cabeçalho do `frame` 21
Quadro 6 - Valores sensoriados 21
Quadro 7 - Algoritmo de treinamento 23
Quadro 8 - Entradas e saídas utilizadas no treinamento da RNA 23
Quadro 9 - Treinamento, aplicação soma ponderada e função de transferência 24
Quadro 10 - Comparação das características dos trabalhos correlatos 31
Quadro 11 - Detalhamento do caso de uso `UC01 - Cadastrar sinal` 33
Quadro 12 - Detalhamento do caso de uso `UC02 - Cadastrar amostra de sinal` 34
Quadro 13 - Detalhamento do caso de uso `UC03 - Treinar rede neural artificial` 34
Quadro 14 - Detalhamento do caso de uso `UC04 - Reconhecer sinal` 34
Quadro 15 - Preenchimento conjunto de coordenadas da amostra, tokens 41
Quadro 16 - Insere valor no conjunto de coordenadas 42
Quadro 17 - Código fonte treinamento RNA 44
Quadro 18 - Código fonte treinamento RNA - simular 45
Quadro 19 - Código fonte treinamento RNA - ajuste pesos sinapses 45
Quadro 20 - Código fonte treinamento RNA - erros saída 46
Quadro 21 - Código fonte treinamento RNA - propagação do sinal 46
Quadro 22 - Código fonte treinamento RNA - retropropagando o erro 47
Quadro 23 - Classificar amostra de sinal 49

LISTA DE TABELAS

Tabela 1 - Tabela de resultados do reconhecimento de sinais estáticos e dinâmicos.....55
Tabela 2 - Tabela de resultados do reconhecimento de sinais estáticos.....57
Tabela 3 - Tabela de resultados do reconhecimento de sinais dinâmicos 58

LISTA DE ABREVIATURAS E SIGLAS

3D – Três Dimensões

API – *Application Programming Interface*

EA – Enterprise Architect

HTML5 – Hypertext Markup Language, versão 5

INES – Instituto Nacional de Educação de Surdos

JSON – *JavaScript Object Notation*

LIBRAS – Língua Brasileira de Sinais

MEC – Ministério da Educação

RF – Requisito Funcional

RNF – Requisito Não-Funcional

RNA – Rede Neural Artificial

SGBD – Sistema Gerenciador de Banco de Dados

UC – Caso de Uso

UML – *Unified Modeling Language*

USB – *Universal Serial Bus*

XML – *Extensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO 14
1.1 OBJETIVOS 15
1.2 ESTRUTURA 15
2 FUNDAMENTAÇÃO TEÓRICA 16
2.1 LIBRAS 16
2.2 LEAP MOTION 18
2.3 RECONHECIMENTO DE PADRÕES 22
2.4 REDES NEURAIS COM O ALGORITMO DE *BACKPROPAGATION* 22
2.5 TRABALHOS CORRELATOS 24
2.5.1 Ferramenta para transcrição do alfabeto datilológico para texto utilizando Microsoft Kinect 25
2.5.2 Ferramenta de reconhecimento de gestos da mão 26
2.5.3 Dicionário LIBRAS 29
2.5.4 Comparação dos trabalhos correlatos 31
3 DESENVOLVIMENTO DO PROTÓTIPO 32
3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO 32
3.2 ESPECIFICAÇÃO 32
3.2.1 Diagramas de casos de uso 32
3.2.2 Diagrama de classes 34
3.2.2.1 Diagrama de classe estrutura do sinal 35
3.2.2.2 Diagrama de classe da estrutura de treinamento da RNA e reconhecimento de sinal 36
3.2.2.3 Diagrama de classes da estrutura dos dados 38
3.2.3 Diagrama de atividades 39
3.3 IMPLEMENTAÇÃO 40
3.3.1 Ferramentas utilizadas 40
3.3.2 Implementação cadastro de sinais: recuperação da estrutura do sinal 40
3.3.3 Topologia da RNA 42
3.3.4 Implementação do algoritmo *backpropagation* 43
3.3.5 Treinamento da RNA 47
3.3.6 Reconhecimento de sinais 48
3.3.7 Representação gráfica dos sinais 49

3.3.8 Operacionalidade da implementação 50
3.4 RESULTADOS E DISCUSSÕES 53
3.4.1 Amostras de testes 53
3.4.2 Reconhecimento de sinais 53
3.4.2.1 Experimento 01: reconhecimento de sinais estáticos e dinâmicos 54
3.4.2.2 Experimento 02: reconhecimento de sinais estáticos 56
3.4.2.3 Experimento 03: reconhecimento de sinais dinâmicos 57
4 CONCLUSÕES 59
4.1 EXTENSÕES 59
REFERÊNCIAS 60

1 INTRODUÇÃO

O fato de não ouvir traduz-se em dificuldade no processo de comunicação interativa com os ouvintes, desconhecedores da língua gestual, e na privação de grande parte da informação, veiculada numa sociedade de ouvintes pela língua oral (MARTINS, 2012).

A Língua Brasileira de Sinais (LIBRAS) é uma linguagem que tem adquirido espaço na sociedade por conta dos movimentos surdos em prol de seus direitos. “Atualmente é reconhecido como meio legal de comunicação e expressão a Língua Brasileira de Sinais – LIBRAS e outros recursos de expressão a ela associados” (BRASIL, 2002). Com a promulgação da Lei de Libras, a comunidade surda pôde comemorar mais uma conquista, afinal a mesma estabelece não apenas LIBRAS como linguagem, mas também outros direitos de suma importância para a comunidade surda, tais como: apoio à difusão da linguagem por parte do poder público e empresas concessionárias de serviços públicos, atendimento adequado por parte destes aos portadores de deficiência auditiva, além da inclusão do ensino de LIBRAS nos cursos de formação de Educação Especial, de Fonoaudiologia e de Magistério nas três esferas do executivo (BRASIL, 2002).

Tal como nas linguagens orais-auditivas, em LIBRAS existem palavras que são os sinais. Em LIBRAS existem cinco parâmetros para formação de um sinal: a configuração de mão diz respeito à forma da mão; a orientação da palma é a direção que a palma da mão aponta na realização do sinal; a locação refere-se ao lugar onde o sinal será executado; o movimento, que pode ou não estar presente nos sinais; os marcadores não manuais são as expressões faciais (GESSER, 2009). Como em qualquer outra linguagem, existem dificuldades para compreensão dos sinais, isso porque, conforme Tavares e Carvalho (2011), LIBRAS é dotada de uma gramática própria e possui os níveis linguístico, fonológico, morfológico, sintático, semântico e pragmático.

Computacionalmente para reconhecer um sinal, é necessário o auxílio de um mecanismo para digitalizá-lo para posterior interpretação. A digitalização de um gesto pode ser realizada de diversas formas, utilizando mídias como fotografias, vídeos e conjunto de vértices. Existem diversos dispositivos de sensoriamento que possuem dentre suas funcionalidades o reconhecimento de gestos, esse é o caso do Microsoft Kinect da empresa Microsoft (KINECT, 2015), Wii Remote da empresa Nintendo (NINTENDO, 2015) e do Leap Motion da empresa homônima (LEAP MOTION, 2015), dentre outros. Dentre os dispositivos mencionados, destaca-se o Leap Motion que é um dispositivo sensorial capaz de captar os movimentos dos dedos e mãos com precisão milimétrica.

Diante do exposto, com o objetivo de propiciar uma maior integração entre ouvintes e surdos e a fim de ajudar a garantir os direitos estabelecidos à comunidade surda pela Lei de Libras (BRASIL, 2002), este trabalho propôs o desenvolvimento de um protótipo para testar a capacidade de identificar sinais em LIBRAS através da utilização do dispositivo Leap Motion como mecanismo de captura para posterior interpretação e textualização dos mesmos.

1.1 OBJETIVOS

Este trabalho tem como objetivo disponibilizar um protótipo que seja capaz de identificar e textualizar alguns sinais realizados em LIBRAS.

Os objetivos específicos são:

a) utilizar o dispositivo Leap Motion para sensoriar/capturar os sinais feitos pelo usuário;
b) reconhecer os sinais feitos pelos usuários, transcrevendo-os para a forma textual a partir do uso de uma RNA com o algoritmo *backpropagation.*

1.2 ESTRUTURA

O trabalho foi dividido em quatro capítulos: no primeiro, consta uma introdução sobre o assunto e os objetivos. O segundo capítulo apresenta a fundamentação teórica necessária para o desenvolvimento do protótipo, passando pela conceituação da linguagem LIBRAS, apresentação do dispositivo de sensoriamento Leap Motion, abordagem do algoritmo *backpropagation* e trabalhos relacionados. O terceiro capítulo traz informações sobre o desenvolvimento do trabalho, requisitos, especificação, implementação, operacionalidade e resultados. No quarto capítulo são relatadas as conclusões do trabalho e suas possíveis extensões e melhorias.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo tem como objetivo explorar os principais assuntos necessários para realização deste trabalho. Os assuntos foram subdivididos em quatro partes. A seção 2.1 conceitua a linguagem LIBRAS. A seção 2.2 aborda o dispositivo de sensoriamento Leap Motion. Na seção 2.3 é apresentado o algoritmo *backpropagation*. Por fim, na seção 2.4 são descritos alguns trabalhos correlatos.

2.1 LIBRAS

Conforme Santana (2007, p. 33),

> Conferir a língua de sinais o estatuto de língua não tem apenas repercussões linguísticas cognitivas, mas também sociais. Se ser anormal é caracterizado pela ausência de língua e de tudo que ela representa (comunicação, pensamento, aprendizagem etc), a partir do momento em que se tem a língua de sinais como língua do surdo, o padrão de normalidade também muda. Ou seja, a língua de sinais legítima o surdo como "sujeito de linguagem" e é capaz de transformar a "anormalidade" em diferença. Isso é resultado de uma luta pela redefinição do que é considerado normal. A ideia de que surdez é uma diferença traz com ela uma delimitação de esferas sociais: a identidade surda, a cultura surda, a comunidade surda.

Não é de se estranhar que a visão da surdez seja tratada por muitos como uma diferença linguística e cultural. Pode-se observar isso quando nos depara-se com as definições de língua, línguas de sinais e LIBRAS utilizadas pela Secretaria de Educação Especial do Ministério da Educação (MEC) [1].

Quanto aos aspectos da linguagem, conforme é exposto por Gesser (2009), LIBRAS é composta por sinais, que por sua vez são formados a partir da combinação dos movimentos da mão em um determinado formato e lugar, podendo ser uma parte do corpo ou em frente ao corpo.

Para formar sinais em LIBRAS são utilizados os parâmetros:

a) configuração de mão: são formas das mãos, que podem ser datilológicas (alfabeto manual) ou formas feitas. A Figura 1 representa o sinal `aprender`;

[1] Língua é um sistema de signos compartilhado por uma comunidade linguística comum. A fala ou os sinais são expressões de diferentes línguas. Línguas de sinais são línguas que são utilizadas pelas comunidades surdas e apresentam propriedades específicas das línguas naturais, sendo, portanto, reconhecidas enquanto línguas pela Linguística. As línguas de sinais são visuais-espaciais captando as experiências visuais das pessoas surdas. LIBRAS é a Língua Brasileira de Sinais (SECRETARIA DE EDUCAÇÃO ESPECIAL, 2004).

Figura 1 - Sinal referente a aprender em LIBRAS

Fonte: Amo fazer atividades (2015).

b) locação: é o ponto onde incide a mão predominante configurada, podendo tocar alguma parte do corpo ou estar em um espaço neutro à frente do corpo. A Figura 2 representa o sinal `importante`, onde o ponto de articulação é logo à frente da cabeça;

Figura 2 - Sinal referente a importante em LIBRAS

Fonte: Amo fazer atividades (2015).

c) movimento: os sinais podem ter movimento ou não. A Figura 1 e a Figura 2, que representam os sinais *`aprender`* e `importante`, respectivamente, utilizam o parâmetro movimentação;

d) orientação da palma: os sinais têm uma direção com relação aos parâmetros das alíneas a), b) e c). Pode-se observar na Figura 3 o uso do parâmetro para formar o sinal `comunicar`;

Figura 3 - Sinal referente a comunicar em LIBRAS

Fonte: Amo fazer atividades (2015).

e) marcadores não manuais: muitos sinais têm como traço diferenciador e quantitativo a expressão facial.

Em relação à gramática LIBRAS, ela não pode ser estudada da mesma forma que a Língua Portuguesa falada. Isso se deve ao fato que LIBRAS possui gramática própria que reflete a forma como o surdo processa suas ideias (STROBEL; FERNANDES, 2008). Para demonstrar tal diferença os autores trazem alguns exemplos. Os exemplos podem ser verificados no Quadro 1 e Quadro 2.

Quadro 1 - Estrutura gramatical em LIBRAS: Eu irei para casa

1	LIBRAS...: EU IR CASA
2	Português: Eu irei para casa
3	para - não se usa em LIBRAS porque está incorporado ao verbo

Fonte: adaptado de Strobel e Fernandes (2008, p. 15).

Quadro 2 - Estrutura gramatical em LIBRAS: Quantos anos você tem?

1	LIBRAS...: IDADE VOCÊ (expressão facial de interrogação)
2	Português: Quantos anos você tem?

Fonte: adaptado de Strobel e Fernandes (2008, p. 15).

Strobel e Fernandes (2008) ainda dão ênfase que na estruturação da frase em LIBRAS não são usados artigos, preposições e conjunções. Segundo os autores, esses conectivos estão incorporados ao sinal. Porém, essas características não limitam a capacidade linguística. Pesquisas realizadas com várias línguas de sinais, como a Língua de Sinais Americana e a Língua Brasileira de Sinais, mostraram que tais línguas são muito complexas e apresentam todos os níveis de análises da linguística tradicional. A diferença básica está no canal em que tais línguas expressam-se para estruturar a língua, um canal essencialmente visual (SECRETARIA DE EDUCAÇÃO ESPECIAL, 2004, p. 20).

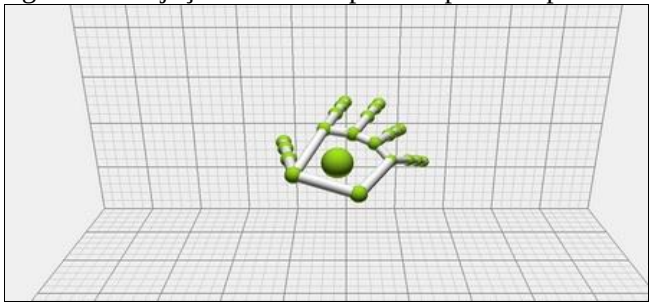
2.2 LEAP MOTION

Segundo Potter, Araullo e Carter (2013, p. 176, tradução nossa):

> O dispositivo Leap Motion é um sensor que visa traduzir os movimentos das mãos em comandos de computador. O dispositivo em si é uma unidade de oito por três centímetros que se conecta ao USB em um computador. O dispositivo é colocado virado para cima em uma superfície onde se deseja capturar o movimento, o sensor detecta a área acima em um intervalo de aproximadamente um metro.

O Leap Motion suporta diversos ambientes de desenvolvimento, tais com JavaScript, Unity / C#, C++, Java, Python e Objetive-C (LEAP MOTION, 2015). No endereço *web* do dispositivo ainda pode ser encontrada a documentação da *Application Programming Interface* (API), o guia de programação para os ambientes suportados, além de exemplos. A Figura 4 representa a projeção de uma mão capturada pelo dispositivo Leap Motion e projetada em um ambiente de três dimensões (3D).

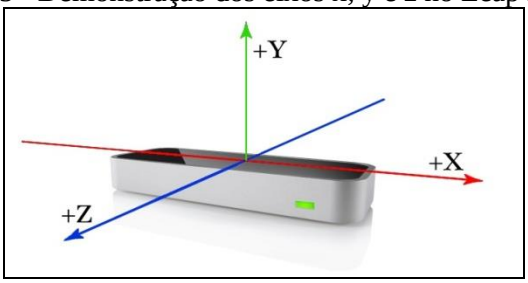
Figura 4 - Projeção de mão capturada pelo Leap Motion



Fonte: Leap Motion (2015).

Na parte de captura, dentre outras funcionalidades, o dispositivo Leap Motion fornece as coordenadas em unidades de milímetros. A representação da ponta de um dedo, por exemplo, é determinada por (x , y , z) = [100 , 100 , -100], tal que, os números são milímetros negativos ou positivos. O centro de referência dos eixos x, y, e z é o próprio dispositivo como pode-se perceber na Figura 5.

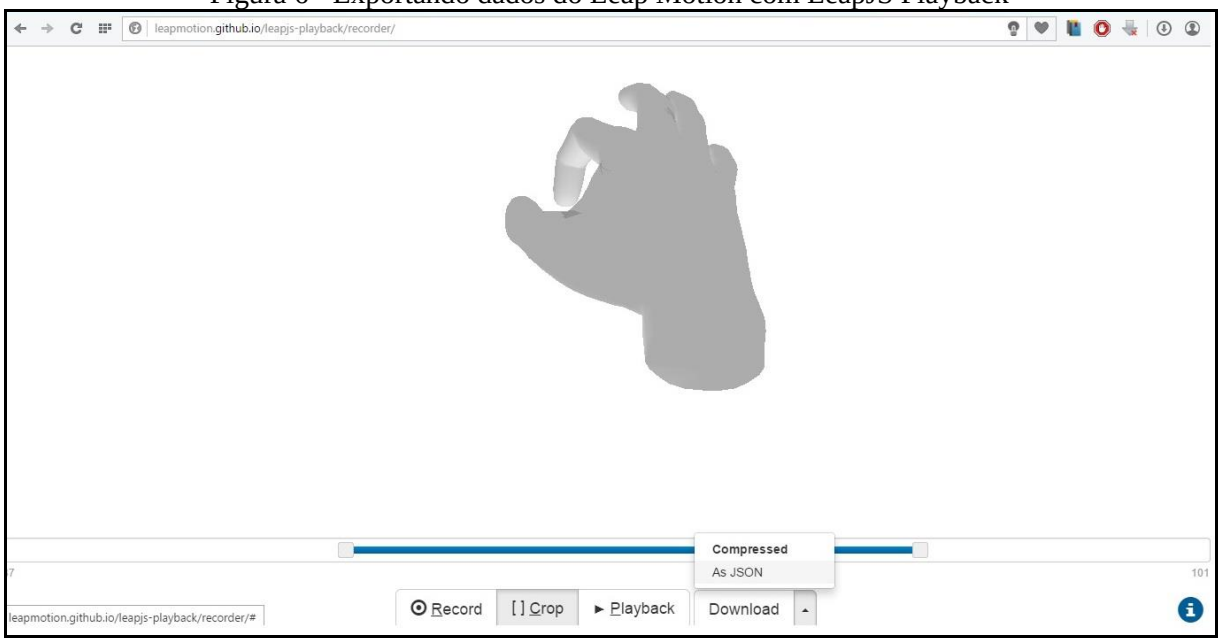
Figura 5 - Demonstração dos eixos x, y e z no Leap Motion



Fonte: Leap Motion (2015).

Para sensoriar o conjunto de coordenadas representadas pelos vértices x, y e z relativos aos movimentos, a empresa Leap Motion disponibiliza o aplicativo LeapJS Playback 0.2.1. Tal aplicativo utiliza bibliotecas JavaScript desenvolvidas pela mesma empresa. Essas bibliotecas são utilizadas para reconhecer o dispositivo e realizar a captura dos dados sensoriados. Desta forma, basta conectar o dispositivo Leap Motion ao computador e a aplicação irá começar a detectar os movimentos realizados dentro da zona de captura do dispositivo. A Figura 6 apresenta a reconstrução do sinal a partir do conjunto de coordenadas sensoriadas pelo dispositivo Leap Motion através da aplicação LeapJS Playback.

Figura 6 - Exportando dados do Leap Motion com LeapJS Playback



Fonte: Leap Motion (2015).

Como o LeapJS Playback é uma aplicação web, a representação gráfica do objeto ocorre sobre um componente `canvas` nativo do Hypertext Markup Language, versão 5 (HTML5). Na Figura 6 pode-se observar os botões `Record`, `Crop`, `Playback` e `Download`. Eles são utilizados para ativar as funcionalidades responsáveis pela inicialização da captura dos dados sensoriados pelo Leap Motion, delimitação do início e o fim dos dados capturados, execução gráfica do conteúdo armazenado e *download* dos dados.

A funcionalidade de *download* permite exportar o conteúdo sensoriado em arquivo texto com extensão no formato `JSON`. O arquivo `JSON` exportado é composto por duas partes principais, são elas: `metadada` e `frames`. O Quadro 3 representa a estrutura do arquivo com as divisões mencionadas.

Quadro 3 - Representação `metadata` e `frame`

```
1  {
2      {
3          "metadata": {...},
4      },
5      "frames": [...]
6  }
```

A parte `metadata` é composta por informações como: versão do formato do arquivo, gerador, quantidade de *frames* contidos, tempo de intervalo de captura dos *frames*, data e hora da modificação e título. O Quadro 4 representa resumidamente as informações contidas no `metadata`.

Quadro 4 – Interior do `metadata`

```
1  "metadata":
2  {
3      "formatVersion": 2,
4      "generateBy": "LeapJs Playback 0.2.1",
5      "frames": 1,
6      ...
7  }
```

A parte dos `frames` é imediatamente subdividida em duas partes, a primeira parte contém o cabeçalho que identifica o escopo e a informação. Nesta, pode-se observar os identificadores (`id` e `timestamp`), informações sobre as mãos (`hands`), dedos (`pointable`) e zona de captura (`interactionBox`). O Quadro 5 representa a estrutura do cabeçalho do `frame`.

Quadro 5 - Cabeçalho do `frame`

```
[
    "id", "timestamp",
    {
        "hands": [...]
    },{
        "pointables": [...]
    },{
        "interactionBox": [...]
    }
]
```

A segunda parte do `frame` (Quadro 6) apresenta uma fração da estrutura em que são armazenados os valores sensoriados.

Quadro 6 - Valores sensoriados

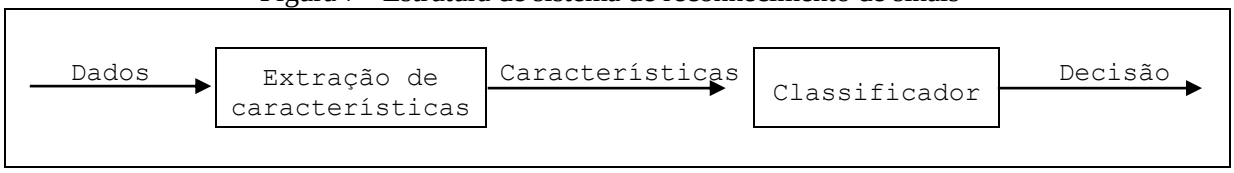
```
[ // 1 grupo para cada frame existente
    247979,1575673823206, //ID (frame), timestamp
    [
        [
            170, // ID(mão)
            "left", // type
            [0.0936169, 0,989947, -0.106025], //direction
            [0.251037, -0,126522, -0.959673], //palmNormal
            //armbases
            [[-55.9116, 128.104, 74.7416, ...]
         ],
    ],
    //pointable - 5 grupos, 1 para cada dedo
    [ [
         1700, // ID(dedo)
         [0.246118, 0,906015, -0.344329], //direction
         170, // HandId
         48.7657, //lenght
         //bases, indica os vértices da falange, 4, grupos
         [[2.48355, 156.152, 108.35], ...]
         ], ... ]

]
```

No Quadro 6 os valores estão relacionados aos identificadores contidos na parte do cabeçalho do `frame`, logo pode-se verificar o valor dos identificadores como `id`, `timestamp`. Os grupos de valores relacionados aos identificadores `hands` e `pointable`, entre outros, aptos a fornecer informações acerca de sua identidade. A segunda parte do `frame` também armazena todos os valores que representam o objeto sensoriado pelo dispositivo Leap Motion através de vértices com valores relativos às coordenadas `x`, `y` e `z` de um plano de três dimensões. Os valores dos vértices disponibilizados são relativamente precisos e permitem representar computacionalmente o objeto sensoriado. Também é importante destacar que ao exportar um gesto com movimento, a sequência do movimento é dividida em diversos `frames`, sendo que cada `frame` contém os dados sensoriados ao longo da linha do tempo.

2.3 RECONHECIMENTO DE PADRÕES

A estrutura clássica de um sistema de reconhecimento de padrões é constituída por dois blocos: um bloco de extração de características, também designada por padrões, e um classificador. O primeiro bloco é responsável por selecionar a informação relevante para a decisão, transformando os dados providos pelos sensores em um conjunto menor de valores denominados características, que posteriormente serão processados pelo segundo bloco, o qual terá como saída a decisão (MARQUES, 2005). A Figura 7 ilustra a estrutura referida.

Figura 7 - Estrutura de sistema de reconhecimento de sinais



Fonte: adaptado de Marques (2005, p. 3).

A extração das características consiste em identificar quais são os parâmetros mais relevantes e separá-los em um conjunto menor para a classificação. Contudo, em casos em que as características são um conjunto reduzido, não há necessidade de tal processo, sendo que todas serão consideradas na classificação. A escolha das características não possui uma metodologia que possa ser aplicada de forma genérica. Isso, segundo Marques (2005), se deve ao fato que este processo depende do contexto do problema, da compreensão dos fenômenos envolvidos e da fonte da informação (sensores diferentes).

O classificador, ao contrário da extração de características é independente da natureza do problema. Em consequência dessa diferença, é possível utilizar técnicas de classificação baseadas em métodos estatísticos como redes neurais para classificação em escopos genéricos, tais como, reconhecimento de fala, processamento de sinais de radar, detecção de avarias, entre outras (MARQUES, 2005).

2.4 REDES NEURAIS COM O ALGORITMO DE *BACKPROPAGATION*

Segundo Tissot, Camargo e Pozo (1996, apud SHANG; BENJAMIN, 2012), Redes Neurais Artificiais (RNA) são frequentemente utilizadas para detectar tendências. Os autores ainda destacam que RNAs são especialmente usadas para encontrar soluções genéricas para problemas onde um padrão de classificação precisa ser extraído.

Ainda segundo Tissot, Camargo e Pozo, (1992, apud OOYEN; NIENHUIS, 2012), o procedimento de treinamento *backpropagation* utiliza vetores que mapeiam um conjunto de entradas para um conjunto de saídas. Durante o processo de aprendizagem da RNA, um vetor de entrada é apresentado para rede e propagado para determinar os valores da saída. Em

seguida, o vetor de saída produzido pela rede é comparado com o vetor de saída desejado, resultando num sinal de erro. Este processo é repetido até que a rede responda, para cada vetor de entrada, com um vetor de saída com valores suficientemente próximos dos valores de saída desejados. Tissot, Camargo e Pozo (1992, apud OOYEN; NIENHUIS, 2012) ainda indicam que o algoritmo de treinamento *backpropagation* é o mais utilizado para fazer o reconhecimento de sinais. O Quadro 7 apresenta os passos necessários para o treinamento.

Quadro 7 - Algoritmo de treinamento

```
1° passo: arbitrar pesos para as conexões entre a camada de entrada e a
unidade sigma

2° passo: aplicar um padrão de sinais de entrada Xj e calcular a soma
ponderada S.

3° passo: passar a soma ponderada S para a função de transferência:

   a) se a saída Yj estiver correta, volta ao 2° passo;
   b) se a saída Yj estiver errada e for 0, adicionar cada peso das
      conexões com os sinais de entrada relativo a elas;
   c) se a saída Yj estiver errada e for 1, subtrair de cada peso das
      conexões os sinais de entrada relativo a elas.

4° passo: voltar ao 2° passo.
```

Fonte: adaptado de Tafner, Xerez e Rodrigues Filho (1996, p. 115).

Tafner, Xerez e Rodrigues Filho (1996) demonstram um pequeno exemplo de treinamento da RNA. No exemplo a mesma foi treinada para classificar um grupo de entrada contendo jogadores de futebol e pilotos de automobilismo. O Quadro 8 representa como foram definidos os conjuntos de entrada e saída utilizados no treinamento da RNA.

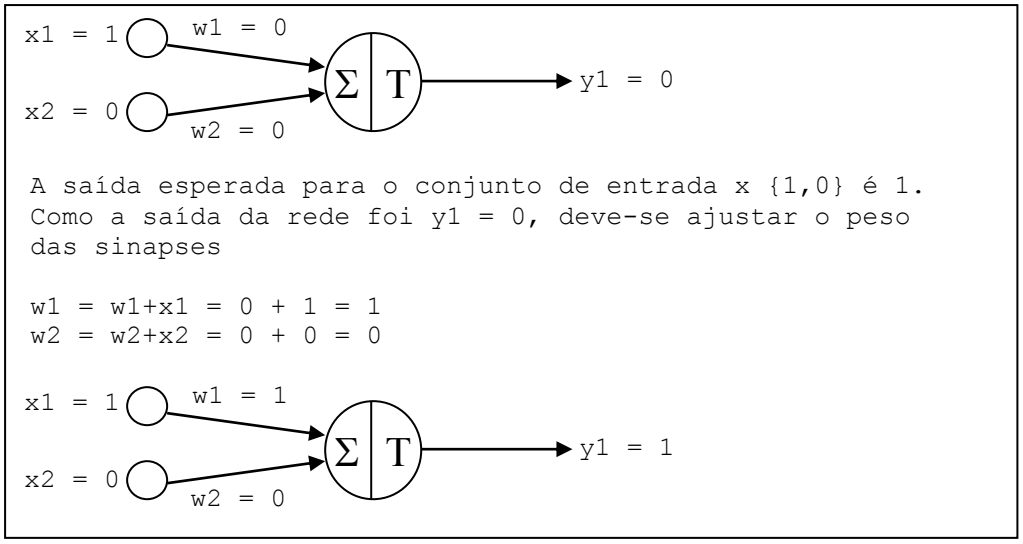
Quadro 8 - Entradas e saídas utilizadas no treinamento da RNA

		0	1
		Futebol	Automobilismo
00	Pelé	X	
01	Zico	X	
10	Emerson		X
11	Piquet		X

Fonte: adaptado de Tafner, Xerez e Rodrigues Filho (1996, p. 116).

No caso para as entradas `00 (Pelé)` e `01 (Zico)`, a RNA deve ter como saída `0 (Futebol)` e para as entradas `10 (Emerson)` e `11 (Piquet)`, a saída deve ser `1 (Automobilismo)`. No exemplo, os autores optaram por inicializar os pesos zerados, no que diz respeito a 1º passo demonstrado no Quadro 7. Em seguida, executou-se os passos 2 e 3 conforme demonstrado no Quadro 9.

Quadro 9 - Treinamento, aplicação soma ponderada e função de transferência



Fonte: adaptado de Tafner, Xerez e Rodrigues Filho (1996, p. 119).

O Quadro 9 demonstra o procedimento executado para treinar a RNA para o elemento de entrada relativo ao piloto `Emerson`. Como entrada para RNA foram definidos os valores `x1 = 1` e `x2 = 0`. Os pesos das sinapses são representados pelas variáveis `w1 = 0` e `w2 = 0`. O valor da saída esperada para entrada foi `1(Automobilismo)`. O símbolo `Σ` representa a função matemática que realiza a soma ponderada dos impulsos sofridos pelo neurônio multiplicados pelo peso das sinapses, ou seja, `Σ = (x1*w1) + (x2*w2) = 0`. O símbolo `T` representa a função binária de ativação do neurônio, retorna `y`, tal que `y = 0 se Σ < 1`, senão `y = 1`. Na primeira execução do algoritmo, a RNA respondeu com a saída `y1 = 0 (Futebol)` para a entrada, como a saída esperada era `1 (Automobilismo)` os pesos das sinapses foram ajustados, `w1` passou a valer `1` e `w2` permaneceu inalterado. Com os novos pesos, a RNA passou a responder com valor correto `y1 = 1 (Automobilismo)`. Esse procedimento foi realizado para todos os elementos do conjunto de entrada (`00, 01, 10 e 11`), ajustando os pesos de forma que a saída da RNA estivesse de acordo com o valor de saída esperado. O processo foi repetido até que a RNA respondesse conforme a saída esperada para todos os elementos de entrada. Ao final, o exemplo de RNA demonstrado pelos autores era capaz de classificar os atletas por modalidade de esporte.

2.5 TRABALHOS CORRELATOS

Nesta seção são abordados três trabalhos correlatos. Na seção 2.5.1 é abordado o trabalho de Santin (2013). A seção 2.5.2 apresenta o trabalho de Bambineti (2009) e, por fim, a seção 2.5.3 aborda o Dicionário LIBRAS disponibilizado pela sociedade Acessibilidade

Brasil (ACESSIBILIDADE BRASIL, 2015). A seção 2.5.4 realiza um comparativo entre os correlatos abordados.

2.5.1 Ferramenta para transcrição do alfabeto datilológico para texto utilizando Microsoft Kinect

O trabalho de Santin (2013) tinha como objetivo desenvolver um interpretador de sinais que fosse capaz de transcrever o alfabeto datilológico ou alfabeto manual para texto. Para alcançar seu objetivo, Santin (2013) utilizou para capturar os gestos o dispositivo de sensoriamento Microsoft Kinect em conjunto com as APIs de `stream` de profundidade contidas no Kinect Windows SDK. Após a captura do gesto, a informação era processada utilizando um algoritmo para simplificar seguimentos de retas, `Ramer-Dougas-Peucker`, resultando em um conjunto de vértices. Este conjunto foi utilizado como entrada em outro algoritmo, desta vez um algoritmo de ordenação, o algoritmo da `tartaruga`. Com esta combinação, tornou-se possível capturar o gesto, simplificá-lo e identificá-lo, comparando os ângulos dos vértices com a informação previamente cadastrada na base de dados. O protótipo utilizava um arquivo XML para armazenar os dados dos sinais. Na Figura 8 é apresentado o exemplo do resultado alcançado na detecção do símbolo datilológico referente à letra `Y`.

Figura 8 - Detecção do símbolo Y

Fonte: Santin (2013).

Santim (2013) realizou dois experimentos. O primeiro consistia em testar o reconhecimento dos sinais isoladamente, ou seja, apenas um sinal era cadastrado na base de dados do protótipo e, em seguida, realizada tentativa de reconhecimento. O segundo consistiu em realizar o reconhecimento dos sinais em uma base contendo todos os símbolos

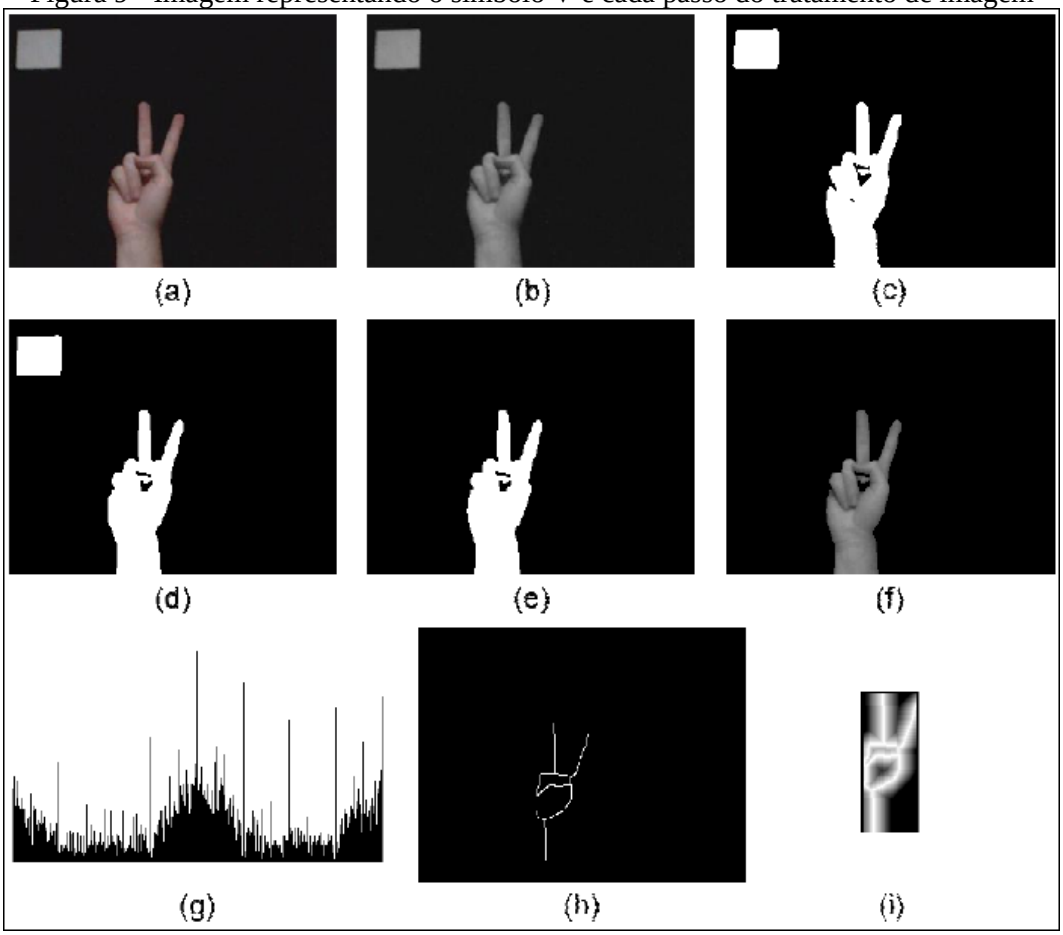
cadastrados. Os resultados alcançados pelo trabalho foram satisfatórios para detecção das letras do alfabeto datilológico no primeiro cenário de testes com uma taxa de acerto próxima a 100%. Porém, nos testes os símbolos ‘H’, ‘J’, ‘K’, ‘W’, ‘X’ e ‘Z’ foram excluídos dos testes por necessitarem de movimento na execução ou por possuírem ângulos muito semelhantes a outros símbolos. A segunda etapa de testes revelou uma taxa de acerto de aproximadamente 12%. Santin (2013) justificou o baixo percentual demonstrando que a chance de existirem ângulos correspondentes torna-se muito alta quando existem muitos sinais na base. O autor ainda destaca que seria possível parametrizar a tolerância utilizada parar calcular os ângulos e assim aprimorar a detecção dos símbolos.

2.5.2 Ferramenta de reconhecimento de gestos da mão

O trabalho de conclusão de curso de Bambineti (2009) teve como objetivo realizar o reconhecimento de gestos em LIBRAS. Para atingir o seu objetivo, Bambineti (2009) utilizou-se uma câmera para capturar uma imagem do gesto realizado. Na imagem contendo a expressão do gesto, após sua binarização foi aplicado o algoritmo de `Zhang-Suen` que processa a imagem de forma a resultar em um esqueleto contendo as características essenciais do gesto expressado. Após o processo de esqueletonização, a ferramenta busca em uma base de dados com amostras previamente cadastradas, um gesto equivalente.

Nos casos em que uma amostra equivalente era encontrada, atingia-se o objetivo, pois o gesto estava identificado. Na Figura 9 são apresentadas as fases de processamento do símbolo datilológico `v`.

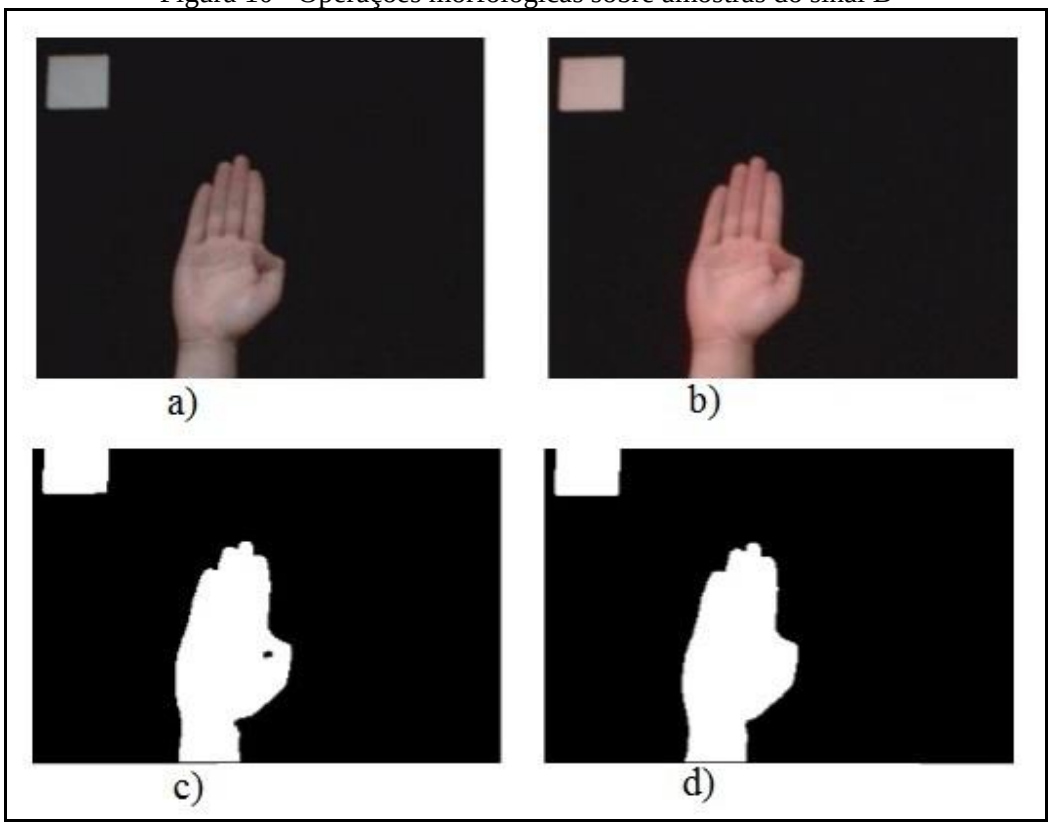
Figura 9 - Imagem representando o símbolo V e cada passo do tratamento de imagem



Fonte: Bambineti (2009).

Durante a etapa de implementação, alguns novos requisitos foram definidos (fundo uniforme, luminosidade, manter distância,) para tornar o ambiente de captura dos gestos controlado. Após a definição dos novos requisitos, os resultados obtidos foram satisfatórios, na faixa de 90%. Os novos requisitos foram justificados por Bambineti (2009) com base em testes de luminosidade utilizando histograma de orientação e esqueletonização. Ambos testes foram realizados sobre duas amostras do sinal `B` contendo diferentes luminosidades. Em ambos os casos, as imagens foram previamente processadas utilizando técnicas morfológicas, conforme mostra a Figura 10.

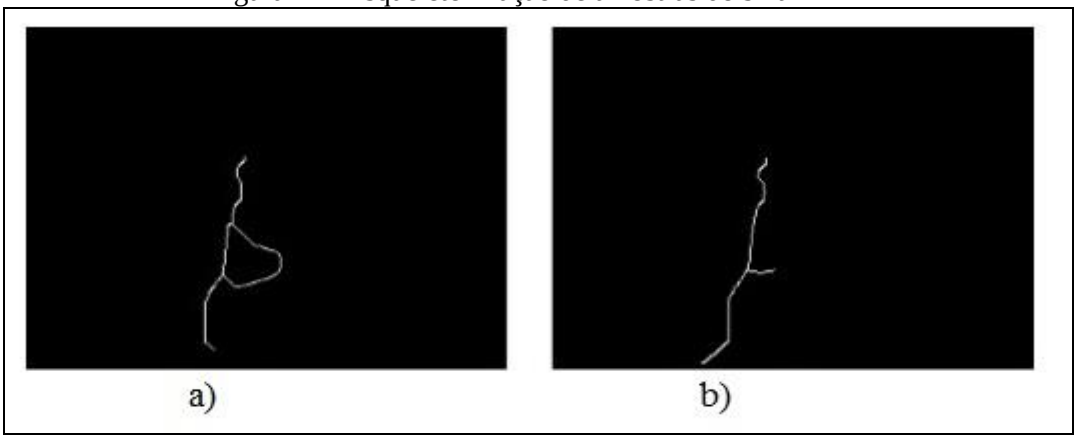
Figura 10 - Operações morfológicas sobre amostras do sinal B



Fonte: Bambineti (2009, p. 53).

Após as operações morfológicas, as imagens começam a apresentar pequenas diferenças estruturais. No teste de esqueletonização a diferença estrutural tornou-se visualmente muito grande, conforme mostra a Figura 11.

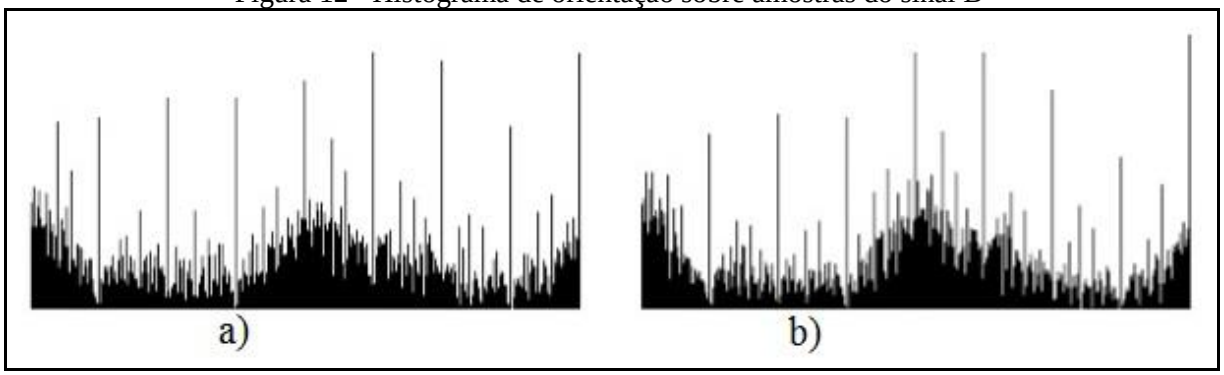
Figura 11 - Esqueletonização de amostras do sinal B



Fonte: Bambineti (2009, p. 53).

Nos testes realizados utilizando histograma de orientação (Figura 12), o autor destacou que as imagens a) e b) geraram histogramas de orientação similares, porém produzindo resultados diferentes. Conforme Bambineti (2009), o histograma da amostra a) possui orientação de 224 graus, enquanto a amostra b) resulta na orientação de 179 graus.

Figura 12 - Histograma de orientação sobre amostras do sinal B



Fonte: Bambineti (2009, p. 54).

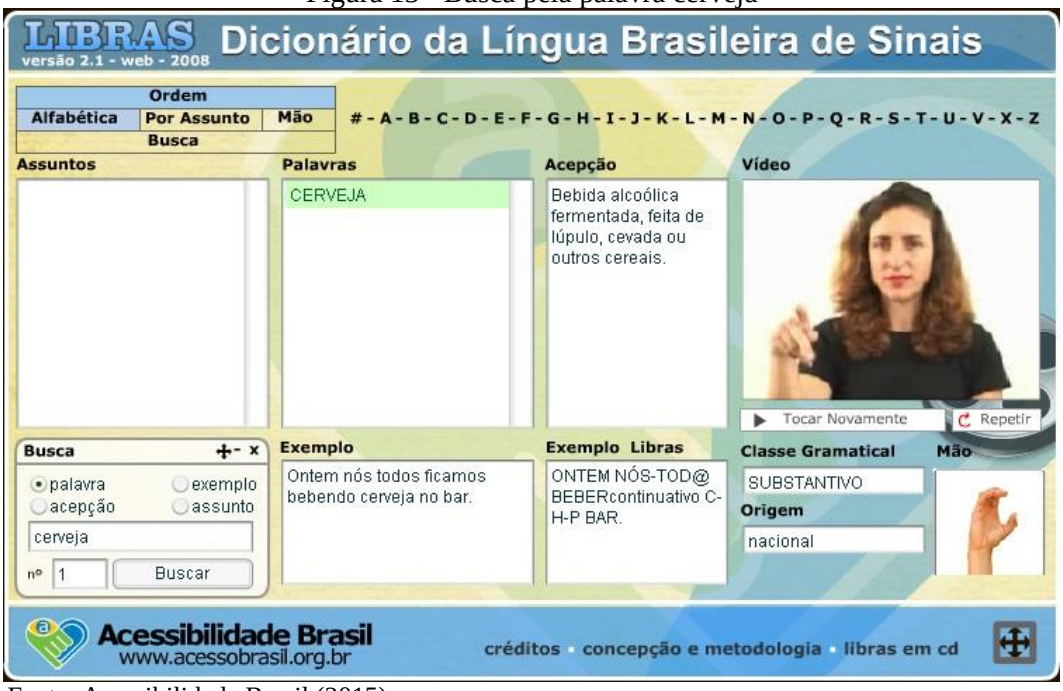
Nos testes o autor identificou que pequenas variações no ambiente impactavam significativamente na funcionalidade de reconhecimento de sinais, justificando assim a necessidade de controlar o ambiente de captura dos sinais.

2.5.3 Dicionário LIBRAS

O dicionário LIBRAS é uma ferramenta gratuita disponibilizada na internet onde é possível traduzir palavras em português para sinais visuais em LIBRAS, um tradutor. A ferramenta foi desenvolvida pela sociedade Acessibilidade Brasil que é formada por especialistas de diversas áreas do conhecimento (ACESSIBILIDADE BRASIL, 2015).

O dicionário LIBRAS se mostra muito interessante por possibilitar que os usuários que ainda não têm o domínio da linguagem se familiarizem com os sinais. O dicionário atualmente se encontra na versão 2.1. Nesta versão, encontra-se a funcionalidade de busca, onde é possível realizar filtros por palavra, acepção, exemplo ou assunto. A Figura 13 demonstra a funcionalidade realizando uma busca por palavra.

Figura 13 - Busca pela palavra cerveja



Fonte: Acessibilidade Brasil (2015).

Ainda existe a funcionalidade de ordenação, tal que a ordenação pode ser realizada por ordem alfabética, por assunto ou pela configuração da mão. A Figura 14 ilustra a ordenação pela configuração de mão.

Figura 14 - Ordenação pela configuração da mão

Fonte: Acessibilidade Brasil (2015).

Os resultados oferecidos pela ferramenta podem ser considerados satisfatórios, visto que além do vídeo contendo a expressão do sinal em LIBRAS, a ferramenta ainda oferece exemplos de uso, acepção, classe gramatical, origem do sinal e configuração da mão. Outra peculiaridade da aplicação dicionário LIBRAS é a disponibilização de cópias em CD-ROM gratuitamente. Na própria aplicação existe um atalho intitulado `libras em cd`, sendo que tal

atalho redireciona a página web para o site do Instituto Nacional de Educação de Surdos (INES) onde é possível entrar em contato para obter uma cópia.

2.5.4 Comparação dos trabalhos correlatos

Esta seção busca realizar uma breve comparação das principais características dos trabalhos correlatos mencionados no decorrer da seção 2.5. O Quadro 10 apresenta tais comparações.

Quadro 10 - Comparação das características dos trabalhos correlatos

Características	Santin (2013)	Bambineti (2009)	Acessibilidade Brasil (2015)
Tradutor	X	X	X
Sinais com movimentos			X
Permite cadastrar sinais	X	X	
Possui representação gráfica do sinal	X	X	X
Stream de imagem	X	X	X
Microsoft Kinect	X		

As características escolhidas para comparar os trabalhos correlatos foram definidas visando encontrar pontos comuns entre os trabalhos correlatos e pontos comuns com o protótipo a ser disponibilizado neste trabalho. A característica tradutor é comum entre todos os correlatos, os trabalhos de Santin (2013) e Bambineti (2009) traduzem sinais para texto, enquanto que o aplicativo da Acessibilidade Brasil (2015) faz o caminho inverso, traduzindo texto para sinais. Dentre os correlatos, apenas o trabalho da Acessibilidade Brasil (2015) tem a capacidade de trabalhar com sinais com movimento, assemelhando-se ao protótipo proposto. Os trabalhos de Santin (2013) e Bambineti (2009) disponibilizam ao usuário final a possiblidade de cadastrar sinais na base. O primeiro através do XML de controle dos dados e o segundo adicionando imagens ao banco de amostras. Já o trabalho da Acessiblidade Brasil (2015) não possui tal característica. Por fim, são listadas as características *stream* de imagem e Microsoft Kinect, todas relacionadas às tecnologias utilizadas pelas aplicações.

3 DESENVOLVIMENTO DO PROTÓTIPO

Este capítulo tem como objetivo demonstrar os passos seguidos no desenvolvimento do protótipo. Os passos foram divididos em quatro partes. Na seção 3.1 são apresentados os requisitos funcionais e não funcionais. A seção 3.2 trata da especificação do protótipo sendo abordados três diagramas, casos de uso, diagrama de classe e diagrama de atividade. A seção 3.3 aborda a implementação, onde são demonstradas as técnicas utilizadas. Por fim, na seção 3.4 são abordados os resultados obtidos, discussões e sugestões de melhorias.

3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

Para atender as necessidades do protótipo foram identificados alguns requisitos funcionais e não funcionais, são eles:

a) utilizar a ferramenta LeapJS Playback para capturar/gravar os sinais do LeapMotion (requisito não-funcional - RNF);
b) identificar os sinais capturados na linguagem LIBRAS utilizando o algoritmo de treinamento *backpropagation* (requisito funcional - RF);
c) disponibilizar uma interface onde seja possível acessar às funcionalidades do protótipo, criar e alterar sinais capturados, treinar a RNA, reconhecer sinais e visualizar de forma textual o significado do sinal feito pelo usuário (RF);
d) ser implementado utilizando a linguagem de programação C++ (RNF);
e) utilizar *Framework .NET* 4.6 para implementar as telas do protótipo (RNF);
f) utilizar *Internet Information Service* 8 para hospedar páginas `web` contendo representação gráfica dos sinais (RNF);
g) utilizar Internet *Explorer* 11 para representar componentes `web` (RNF).

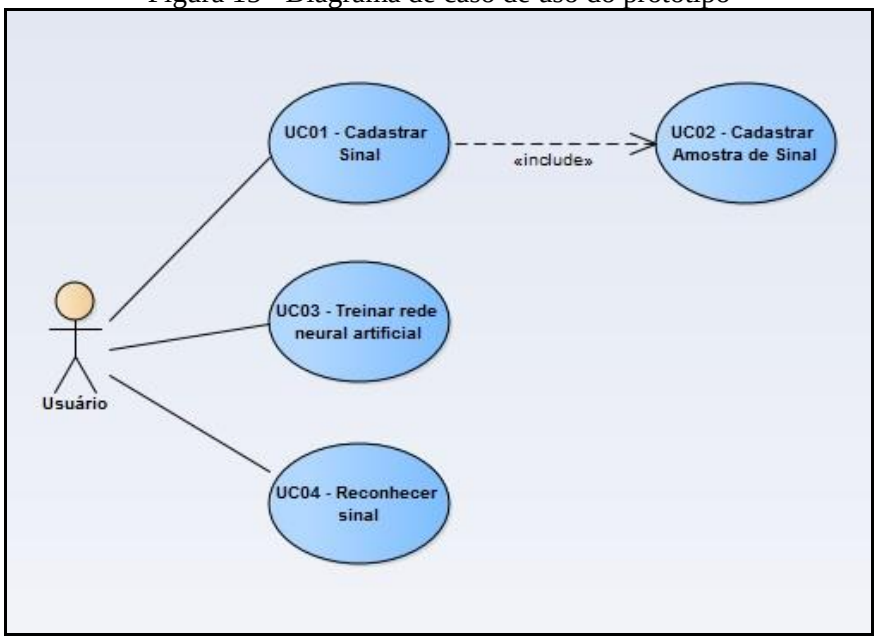
3.2 ESPECIFICAÇÃO

A especificação foi desenvolvida seguindo a metodologia *Unified Modeling Language* (UML) na elaboração dos diagramas de casos de uso, diagrama de classe e diagrama de atividades. Para tal foi utilizada a ferramenta Enterprise Architect (EA).

3.2.1 Diagramas de casos de uso

A Figura 15 demonstra o diagrama de caso de uso do protótipo. Nele estão representadas as ações que o ator `Usuário` pode realizar ao iniciar o protótipo.

Figura 15 - Diagrama de caso de uso do protótipo



O caso de uso `UC01 - Cadastrar sinal` (Quadro 11) demonstra como o ator `Usuário` insere, altera e exclui os sinais.

Quadro 11 - Detalhamento do caso de uso `UC01 - Cadastrar sinal`

Número	01
Caso de Uso	Cadastrar sinais
Cenário principal	1. O usuário informa o código do sinal; 2. O usuário informa o significado; 3. O usuário informa a descrição; 4. O usuário clica no botão ++Sinal; 5. O usuário clica no botão Aplicar.
Cenário alterativo 1	Após o passo 03, se o usuário desejar adicionar amostras do sinal. 1.1. O usuário clica no botão Amostras; 1.2. UC02 – Cadastrar amostra de sinal; 1.3. Voltar ao cenário principal.
Cenário alternativo 2	Após o passo 01, se o usuário desejar excluir o sinal. 1.1. O usuário clica no botão --Sinal; 1.2. Voltar ao cenário principal.
Cenário alternativo 3	Após o passo 01, se o código já existe o protótipo entra em modo de alteração. 1.1. O usuário informa os novos valores para o significado e/ou descrição; 1.2. O usuário clica no botão ++Sinal; 1.3. Voltar ao cenário principal.
Cenário de exceção 1	No passo 04, caso o código informado esteja fora do intervalo 1 até 127 será exibida a mensagem “o código deve estar contido no intervalo de 1 até 127”, caso o significado não seja informado, será exibida a mensagem "Informe o significado", caso a descrição não seja informada, será exibida a mensagem “Informe a descrição”.
Pós-condições	Os dados serão persistidos pelo protótipo.

O caso de uso `UC02 - Cadastrar amostra de sinal` (Quadro 12) demonstra os procedimentos necessários para manutenção das amostras do sinal.

Quadro 12 - Detalhamento do caso de uso `UC02 – Cadastrar amostra de sinal`

Número	02
Caso de Uso	Cadastrar amostra de sinal
Cenário principal	1. O usuário clica no botão Escolher Arquivo; 2. O usuário seleciona o arquivo JSON relativo à amostra; 3. O usuário clica no botão ++Amostra; 4. O usuário clica no botão Aplicar.
Cenário alterativo 1	Após o passo 01, se o usuário desejar excluir amostras do sinal. 1.1. O usuário seleciona a amostra na lista de amostras; 1.2. O usuário pressiona a tecla delete; 1.3. O usuário clica no botão Aplicar 1.4. Voltar ao cenário principal.
Cenário de exceção 1	No passo 01, caso o arquivo selecionado não seja uma amostra válida, será exibida a mensagem “Erro ao preencher coordenadas”.
Pós-condições	Os dados são armazenados na lista de amostras do sinal selecionado.

O caso de uso `UC03 – Treinar rede neural artificial` (Quadro 13) demonstra o detalhamento da funcionalidade de treinamento da RNA.

Quadro 13 - Detalhamento do caso de uso `UC03 – Treinar rede neural artificial`

Número	03
Caso de Uso	Treinar rede neural artificial
Cenário principal	1. O usuário informa a quantidade de iterações que o algoritmo *backpropagation* deve ser executado; 2. O usuário clica no botão Treinar.
Pós-condições	Os dados do treinamento são persistidos.

O caso de uso `UC04 – Reconhecer sinal` (Quadro 14) apresenta os passos da funcionalidade de reconhecimento de sinais.

Quadro 14 - Detalhamento do caso de uso `UC04 – Reconhecer sinal`

Número	04
Caso de Uso	Reconhecer sinal
Cenário principal	1. O usuário clica no botão classificar sinal; 2. O usuário seleciona o sinal a ser classificado.
Cenário de exceção 1	No passo 01, caso o arquivo selecionado não seja uma amostra válida, será exibida a mensagem “Erro ao preencher coordenadas”.
Pós-condições	O quadro Sinal escolhido irá representar a forma do sinal escolhido enquanto que o quadro Sinal identificado, se for identificado algum sinal compatível, demonstrará o mesmo.

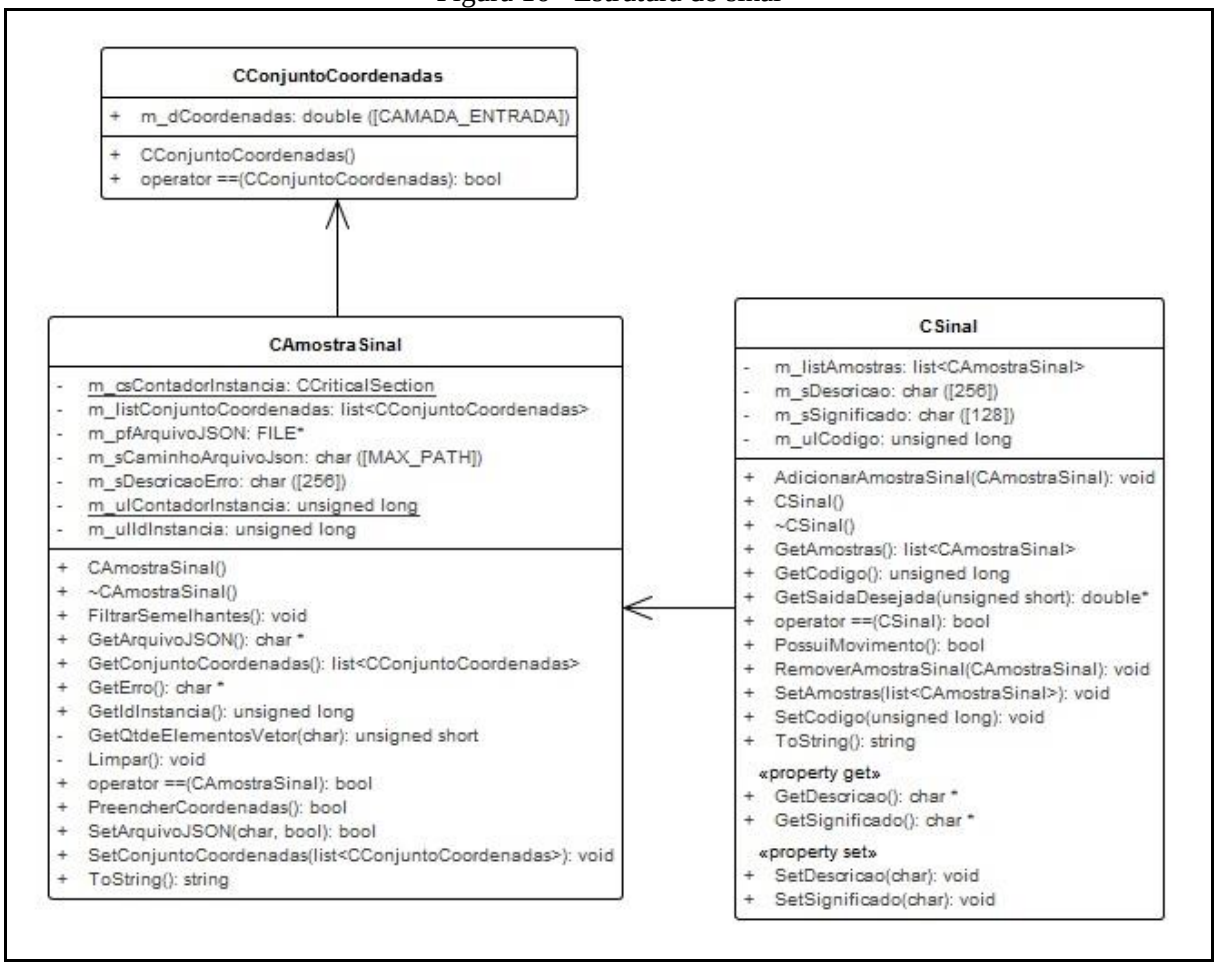
3.2.2 Diagrama de classes

Nesta seção são apresentados os diagramas de classes das principais funcionalidades do protótipo. Também são descritas individualmente as classes presentes no diagrama e suas respectivas responsabilidades no modelo. A seção 3.2.2.1 demonstra a estrutura do sinal; a seção 3.2.2.2 demonstra o diagrama de classe relacionado ao treinamento da RNA e ao reconhecimento dos sinais. Por fim, na seção 3.2.2.3 é demonstrado o diagrama de classes relacionado à manipulação dos dados.

3.2.2.1 Diagrama de classe estrutura do sinal

A Figura 16 apresenta a modelagem das classes responsáveis pela organização das características comuns aos sinais. O modelo é utilizado nas funcionalidades cadastro de sinais, treinamento da RNA e reconhecimento de sinais. No cadastro de sinais, o modelo serve como base para inclusão, alteração e exclusão de sinais e amostras. No treinamento, a modelagem é utilizada para alimentar a entrada da RNA, tal que cada elemento do conjunto de coordenada é uma entrada. A classe `CAmostraSinal` em conjunto com a classe `CBackPropagation`, que será apresentada na seção 3.2.2.2, viabilizam o reconhecimento de sinais.

Figura 16 - Estrutura do sinal



A classe `CSinal` é responsável por armazenar uma lista contendo todas as amostras e propriedades referentes ao sinal. A mesma ainda conta com o método `GetSaidaDesejada` que retorna um vetor de `double` com tamanho fixo de 7 elementos, a mesma quantidade de neurônios na camada de saída da RNA. Os elementos do vetor contêm a representação binária do código do sinal (1 bit por elemento). Essa informação é utilizada no momento do treinamento da RNA e no reconhecimento de sinais. Como a saída desejada é definida com base no código atribuído ao sinal, isso traz certas limitações ao protótipo, visto que a

quantidade de sinais torna-se finita podendo cadastrar no máximo 127 sinais. A limitação está prevista no cadastro de sinais, sendo assim, o protótipo somente aceita o código com valores dentro do intervalo de 1 até 127.

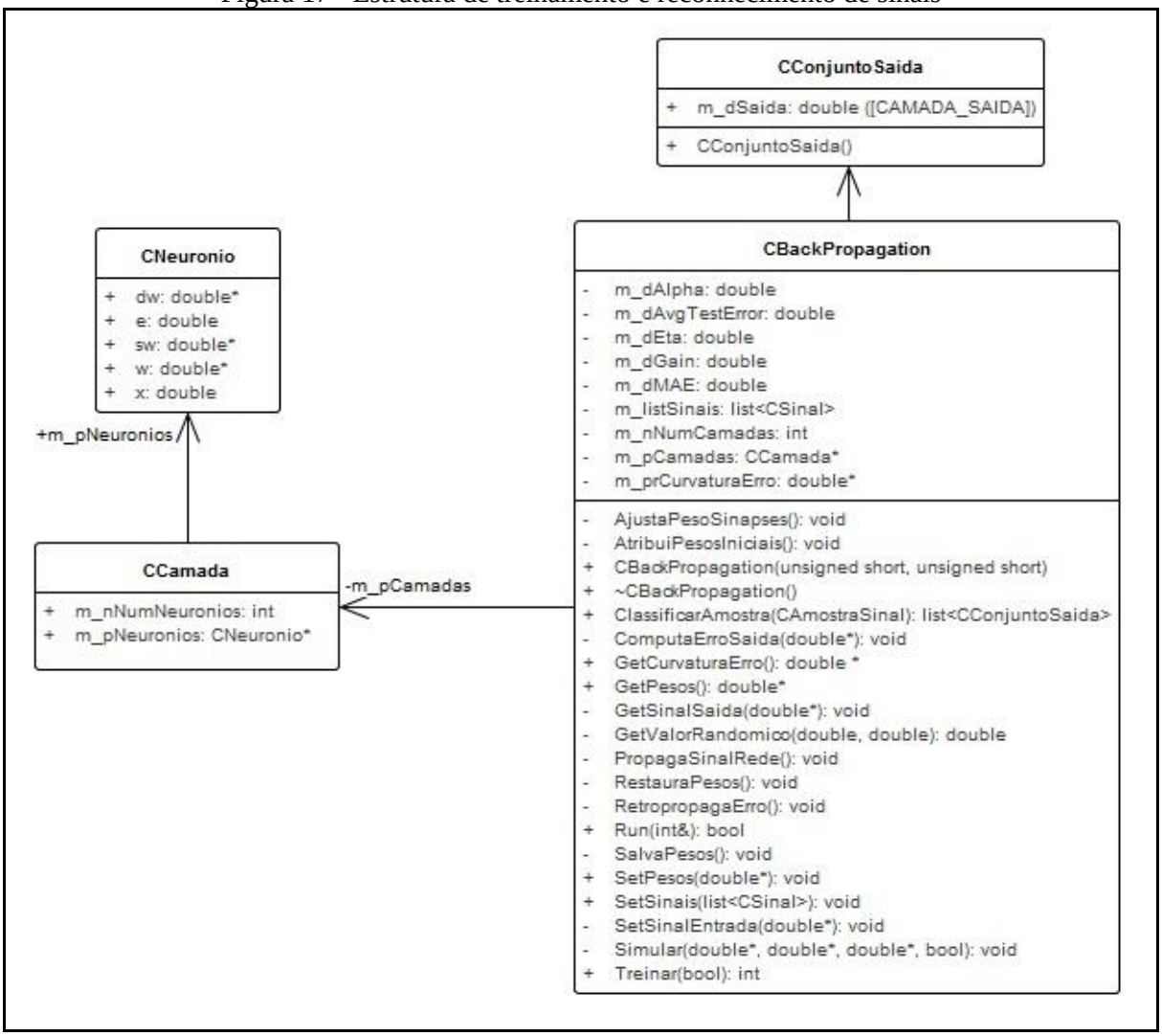
A classe `CAmostraSinal` é composta pelo atributo que define o caminho lógico para arquivo base da amostra, `m_sCaminhoArquivoJson`, a mesma ainda conta com uma lista de conjuntos de coordenadas relativos à amostra. Os atributos estáticos `m_csContadorInstancia` e `m_ulContadorInstancia` são utilizados para diferenciar as instâncias da classe. O atributo `m_pfArquivoJSON` é um ponteiro para o arquivo relativo à amostra. A classe em questão ainda é provida do método `PreencherCoordenadas` que irá popular a lista de conjunto de coordenadas de acordo com os dados relativos à amostra informada. A classe `CConjuntoCoordenada` armazena o valor das coordenadas relativas a cada `frame` da amostra do sinal.

Nota-se que uma instância da classe `CSinal` pode conter `n` instâncias da classe `CAmostraSinal` que por sua vez deve conter uma ou mais instâncias da classe `CConjuntoCoordenadas`.

3.2.2.2 Diagrama de classe da estrutura de treinamento da RNA e reconhecimento de sinal

Na Figura 17 encontra-se a modelagem das classes que compõem a base para o treinamento da RNA e para o reconhecimento de sinais. No diagrama em questão, o papel de protagonista é atribuído à classe `CBackPropagation`, que é responsável pela execução do algoritmo *backpropagation* (ver Figura 7). Esta classe ainda é responsável pelo gerenciamento das classes `CCamada` e `CNeuronio`, utilizadas no processo de treinamento da RNA, e `CConjuntoSaida`, utilizada no processo de reconhecimento de sinais.

Figura 17 - Estrutura de treinamento e reconhecimento de sinais



A classe `CBackPropagation` conta com o método `Run` que inicializa o processo de treinamento, o mesmo recebe por parâmetro a quantidade de iterações que será executado o algoritmo *backpropagation*. O método `Treinar` será utilizado para percorrer todos os conjuntos de coordenadas de todos os sinais cadastrados na base e invocar o método `Simular`. Este por sua vez invocará os métodos `SetSinalEntrada`, `PropagaSinalRede`, `GetSinalSaida`, `ComputaErroSaida`, `RetropropagaErro`, `AjustaPesoSinapses`, todos relativos ao treinamento da RNA. O método `ClassificarAmostra` retorna uma lista contendo instâncias da classe `CConjuntoSaida` que armazena um vetor de `double` com tamanho fixo de 7 elementos relativo à saída da RNA para a amostra recebida na entrada da RNA. A lista retornada serve para identificar o código do sinal identificado. A classe `CBackPropagation` ainda é composta pelos atributos `m_dAlpha`, `m_dAvgTestError`, `m_dEta`, `m_dGain` e `m_dMAE` que representam, respectivamente: o percentual a ser considerado do valor

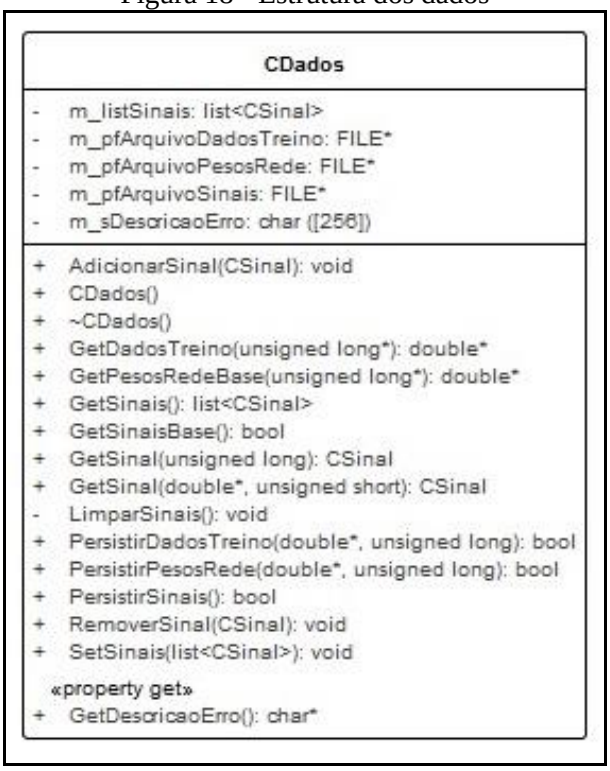
anterior da sinapse, a média de erro da rede após uma iteração, o avanço no ajuste do peso da sinapse, o escopo de variância e o valor médio do erro após treino de um elemento.

As classes `CNeuronio` e `CCamada` fazem parte do algoritmo *backpropagation*. A classe `CBackPropagation` em seu construtor recebe dois parâmetros, um `unsigned short` relativo à quantidade de camadas que serão utilizadas no algoritmo e um vetor de `unsigned short` contendo a quantidade de neurônios por camada. A classe `CNeuronio` é composta apenas por atributos, são eles: `dw`, `e`, `sw`, `w` e `x`. Os atributos mencionados representam, respectivamente a evolução do peso da aresta, o erro em relação à saída esperada, o peso anterior da sinapse, o peso da sinapse e a entrada do neurônio. A classe `CCamada` contém uma lista dos neurônios que estão vinculados à mesma.

3.2.2.3 Diagrama de classes da estrutura dos dados

A Figura 18 demonstra a estrutura da classe `CDados` que foi utilizada para realizar a leitura e escrita dos dados utilizados pelo protótipo. Neste ponto é importante ressaltar que o protótipo não faz uso de um Sistema Gerenciador de Banco de Dados (SGBD), as informações são gravadas em arquivos de texto plano. A estratégia de não utilizar um SGBD foi adotada porque as funcionalidades do protótipo não demandam muita manipulação dos dados, também foi considerada a minimização de dependência de ferramentas de terceiros.

Figura 18 - Estrutura dos dados

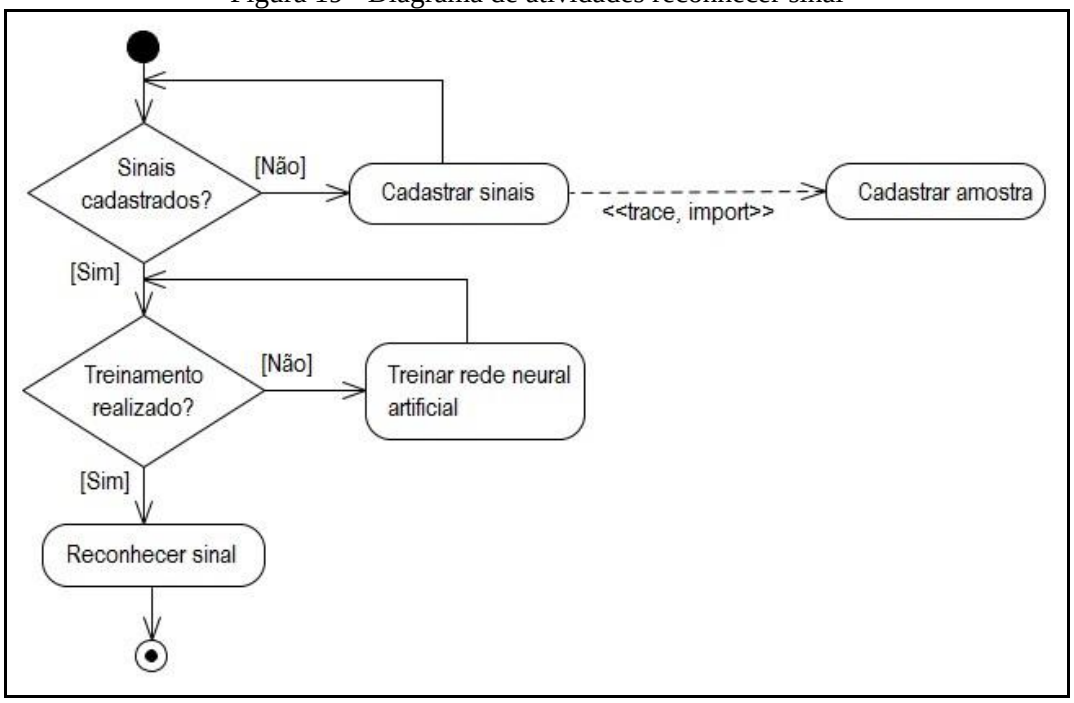


A classe `CDados` contém uma lista de sinais (`m_listSinais`) que é base para execução dos métodos `GetSinaisBase` e `PersistirSinais`. Há três atributos do tipo `FILE*`. Eles são ponteiros para os arquivos que armazenam os sinais cadastrados, os dados do treinamento da RNA e os pesos das sinapses calculados durante o treinamento, que juntos compõem a base de dados do protótipo.

3.2.3 Diagrama de atividades

A Figura 19 demonstra as atividades que devem ser executadas para reconhecer um sinal. Nota-se que as atividades `Cadastrar sinais`, `Cadastrar amostra` e `Treinar rede neural artificial` não são sempre necessárias. O treinamento é necessário apenas se houver a necessidade de incorporar a alterações realizadas no cadastro de sinais à funcionalidade de reconhecimento dos sinais.

Figura 19 - Diagrama de atividades reconhecer sinal



Após iniciar o protótipo pela primeira vez, o usuário deverá realizar o cadastro dos sinais e suas respectivas amostras. Na tela de cadastro de sinais o usuário deverá informar o código, o significado e a descrição do sinal que deseja cadastrar. Ainda na mesma funcionalidade o usuário tem a opção de adicionar amostras do sinal. As amostras cadastradas servirão de entrada para o treino da RNA. Caso não seja informada nenhuma amostra, o sinal não será considerado no treinamento da RNA. Ao terminar a etapa de cadastramento, o usuário deverá realizar o treinamento da RNA para que esta tenha capacidade de classificar e, desta forma, identificar os sinais cadastrados. Ao final do processo, o protótipo persiste os

dados do treino, ou seja, o valor encontrado para as conexões sinápticas. Como os dados do cadastro e do treino são persistidos ao final de seus respectivos processos, ao abrir o protótipo em um segundo momento não há necessidade de realizar tais procedimentos novamente podendo-se executar diretamente a atividade de reconhecimento de sinal. Esse último processo consiste em escolher uma amostra de sinal e apresentar como entrada para a RNA, que irá propagar o sinal utilizando os pesos sinápticos salvos no treinamento gerando uma saída. Essa por sua vez é utilizada para identificar a amostra de sinal escolhida pelo usuário.

3.3 IMPLEMENTAÇÃO

Nesta seção são apresentadas as técnicas e ferramentas utilizadas na etapa de implementação do protótipo.

3.3.1 Ferramentas utilizadas

Na implementação do código fonte do protótipo foi utilizada a ferramenta Microsoft Visual Studio 2013. A mesma disponibiliza um ambiente para desenvolvimento na linguagem C++, linguagem esta utilizada para implementação do protótipo.

3.3.2 Implementação cadastro de sinais: recuperação da estrutura do sinal

No protótipo, o sinal é composto por amostras de sinais, as amostras por sua vez são compostas por conjuntos de coordenadas, as coordenadas são recuperadas a partir das informações presentes nos `frames` contidos no arquivo `JSON`. Como o protótipo não contempla a extração dos dados do Leap Motion, para extrair as amostras dos sinais foi utilizada a aplicação LeapJS Playback 0.2.1.

De posse dos sinais no formato `JSON`, a classe `CAmostraSinal` tem a responsabilidade de popular sua lista de conjunto de coordenadas. O algoritmo utilizado para popular o conjunto de coordenadas divide os dados em `tokens`. Os `tokens` são validados através do método `GetQtdeElementosVetor`, tal que serão considerados apenas os `tokens` que possuírem três elementos no vetor, isso garante que apenas os dados relativos às coordenadas serão considerados, ou seja, todos os dados relativos ao plano 3D são considerados características relevantes para o reconhecimento do sinal. As demais informações são descartadas. O Quadro 15 e Quadro 16 demonstram os códigos desenvolvidos.

Quadro 15 - Preenchimento conjunto de coordenadas da amostra, tokens

```
1  const char separadores[] = { "[]" };
2  psToken = strtok_s(psBuffer, separadores, &psProximoToken);
3
4  if (psToken)
5      psToken = strtok_s(NULL, separadores, &psProximoToken);
6
7  double dCoordenadas[CAMADA_ENTRADA] = { 0 };
8  CConjuntoCoordenadas oConjuntoCoordenadas;
9  unsigned short usTokenContador = 0;
10
11 while (psToken)
12 {
13     usTokenContador++;
14
15     if (usTokenContador % usTokensPorFrame == 0)
16     {
17         if (usContador != CAMADA_ENTRADA)
18         {
19             for (; usContador < CAMADA_ENTRADA; usContador++)
20                 dCoordenadas[usContador] = 0;
21         }
22
23         memcpy(oConjuntoCoordenadas.m_dCoordenadas, dCoordenadas, sizeof(dCoordenadas));
24         m_listConjuntoCoordenadas.push_back(oConjuntoCoordenadas);
```

No Quadro 15, a linha 1 define os delimitadores dos `tokens`, os caracteres `[` e `]`. Após realizar a separação dos `tokens` conforme demonstrado na linha 2, a iteração é inicializada para realizar o preenchimento dos conjuntos de coordenadas. Para recuperar o próximo `token`, é utilizado o comando `strtok_s` passando como parâmetro o valor `NULL` (linha 5). Nos casos em que o conjunto de coordenadas não é inteiramente completado, as demais posições são preenchidas com valor zero, esse tratamento é demonstrado nas linhas 19 e 20.

O Quadro 16 demonstra como os valores são inseridos no conjunto de coordenadas. O teste realizado na linha 1 garante que apenas as informações de coordenadas serão inseridas. Caso a condição do teste seja atendida, os valores das coordenadas `x`, `y`, e `z` são desmembrados (linhas 7 até 12) e inseridos no conjunto de coordenadas conforme demonstrado na linha 24.

Quadro 16 - Insere valor no conjunto de coordenadas

```
if (GetQtdeElementosVetor(psToken) == 3)
{
    memset(&valor, 0, sizeof(valor));

    for (; strlen(psToken); psToken++)
    {
        if (psToken[0] == '.' || psToken[0] == '-' ||
           (psToken[0] >= '0' && psToken[0] <= '9') ||
           ((strlen(psToken) > 1 && (psToken[0] == 'E' || psToken[0] == 'e') &&
           (psToken[1] == '+' || psToken[1] == '-'))))
        {
            valor[strlen(valor)] = psToken[0];
        }
        else if (valor[0])
        {
            if (usContador >= CAMADA_ENTRADA)
            {
                delete[] psBuffer;
                strcpy_s(m_sDescricaoErro,
                "Falha ao preencher vértices da rede. \nQuantidade de vértices maior que o esperado.");
                return false;
            }

            dCoordenadas[usContador] = atof(valor);
            usContador++;

            memset(&valor, 0, sizeof(valor));
        }
    }
```

Após realizar o preenchimento dos dados do conjunto de coordenada da amostra, a estrutura do sinal pode ser persistida na base de dados.

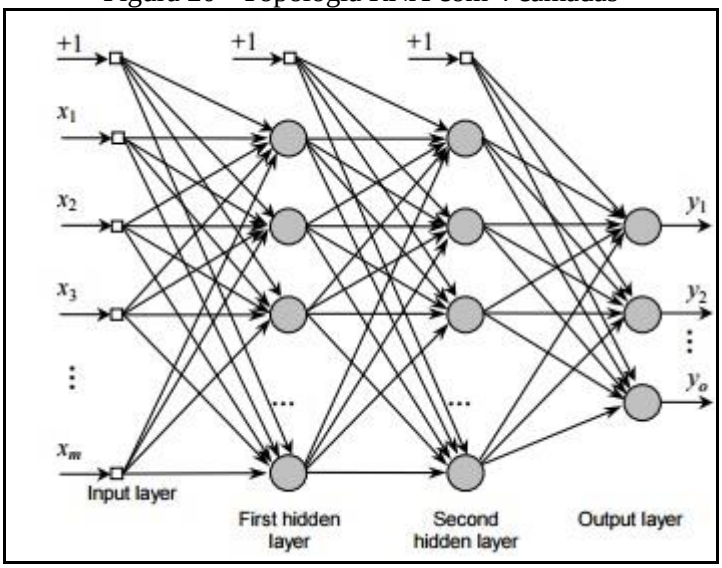
3.3.3 Topologia da RNA

A definição da topologia da RNA foi o processo mais oneroso no desenvolvimento do protótipo. Surgiram problemas de desempenho e ineficiência que exigiram muita atenção. Como não se tratava de uma questão algorítmica e sim de parametrização, o processo de escolha da topologia acabou sendo realizado após diversas tentativas de adequar os dados de entrada ao resultado final. Vale mencionar que a topologia interfere em sua capacidade de generalização e em seu desempenho, pois se trata da arquitetura do classificador da RNA.

Após a realização de diversos treinamentos e análises da capacidade de classificação, optou-se por definir a RNA com 4 camadas, contendo 708 neurônios na camada de entrada, duas camadas ocultas contendo 357 neurônios cada e a camada de saída contendo 7 neurônios. As camadas de entrada e saída não foram definidas com base nos testes, isso porque para contemplar todas as coordenadas relativas ao sinal sensoriado pelo Leap Motion eram necessários no mínimo 708 neurônios, sendo que o número busca satisfazer a demanda para caracterizar gestos compostos pelas duas mãos simultaneamente. A camada de saída foi definida com 7 neurônios porque no decorrer do desenvolvimento do protótipo chegou-se ao consenso que 128 padrões de classificação atenderiam a demanda do protótipo. Diz-se 128 porque cada neurônio da camada de saída representa um bit, sendo possível combiná-los para representar 128 estados. Para evitar confusões vale ressaltar que o cadastro de sinais trabalha

com apenas 127 padrões de classificação, o estado de saída zero (0000000) foi excluído para eliminar o código 0 do cadastro de sinais. Os valores das camadas ocultas foram definidos com base em observações realizadas durante os treinamentos. A Figura 20 demonstra uma RNA com 4 camadas.

Figura 20 - Topologia RNA com 4 camadas



Fonte: Zuben e Attux (2015, p. 24).

Para realizar uma analogia à arquitetura do protótipo, pode-se considerar a Figura 20. Os valores de entrada da RNA são representados pelas variáveis `x1` até `xm`. Os valores constantes `+1` percebidos na camada de entrada e nas camadas ocultas são o viés da RNA. Para representar tal valor foi definida a constante `m_dGain` da classe `CBackPropagation` (Figura 17). Os círculos representam os neurônios contidos em cada camada. Nota-se que na camada de entrada os mesmos foram omitidos, isso ocorre porque a função de ativação somente é aplicada para os neurônios da segunda camada em diante. As setas representam as sinapses, estas realizam a ligação entre os neurônios da RNA. Por fim, os valores `y1` até `yo` representam a saída da RNA.

3.3.4 Implementação do algoritmo *backpropagation*

Para realizar o treinamento da rede neural artificial foi utilizado o algoritmo `Multilayer Perceptron` *backpropagation*. A implementação do algoritmo foi baseada na implementação do algoritmo *Multilayer Perceptron* de Barthelemy (2000). A sequência do Quadro 17 até Quadro 22 demonstram os principais trechos de código que compõem o algoritmo.

O Quadro 17 demonstra parte do método `Treinar` da classe `CBackPropagation`. O código da linha 1 define que serão utilizadas aproximadamente 75% das amostras por sinal.

Esse comportamento é utilizado para tentar prevenir que a RNA memorize os padrões de entrada, além de maximizar o potencial de generalização da mesma. No mais é invocado o método `Simular` na linha 19 para cada conjunto de coordenada pertencente ao sinal.

Quadro 17 - Código fonte treinamento RNA

```
1  hAmostraParaTreino = (unsigned short)(listAmostras.size() * 0.75) + 1;
2
3  for (; itAmostra != itAmostraFinal; itAmostra++, hAmostraParaTreino--)
4  {
5      listConjuntoCoordenadas = itAmostra->GetConjuntoCoordenadas();
6      itConjuntoCoordenadas = listConjuntoCoordenadas.begin();
7      itConjuntoCoordenadasFinal = listConjuntoCoordenadas.end();
8
9      for (; itConjuntoCoordenadas != itConjuntoCoordenadasFinal; itConjuntoCoordenadas++)
10     {
11         dEntradas = itConjuntoCoordenadas->m_dCoordenadas;
12         if (!dEntradas)
13         {
14             delete[] saidaDesejada;
15             delete[] saida;
16             return 0;
17         }
18
19         Simular(dEntradas, saida, saidaDesejada, !bTestar);
20
21         m_dAvgTestError += m_dMAE;
22         count++;
23     }
24
25     if (bTestar && !hAmostraParaTreino)
26         break;
27 }
```

O Quadro 18 apresenta o método `Simular` da classe `CBackPropagation`. Ele é responsável por invocar os métodos que compõem a sequência de execução do algoritmo *backpropagation*. Neste ponto vale mencionar o parâmetro de entrada `bool treinar` (linha 2), o mesmo é necessário pois o método é executado duas vezes, a primeira no processo de ajuste dos pesos sinápticos da RNA e a segunda no processo de testes onde todas as amostras são apresentadas à RNA e a saída deseja comparada aos valores de saída da RNA.

Quadro 18 - Código fonte treinamento RNA - simular

```
1  void CBackPropagation::Simular(double* entrada, double* saida,
2                                 double* saidaDesjada, bool treinar)
3  {
4      if (!entrada)
5          return;
6
7      if (!saidaDesjada)
8          return;
9
10     SetSinalEntrada(entrada);
11
12     PropagaSinalRede();
13
14     GetSinalSaida(saida);
15
16     ComputaErroSaida(saidaDesjada);
17
18     if (treinar)
19     {
20         RetropropagaErro();
21         AjustaPesoSinapses();
22     }
23 }
```

O Quadro 19 apresenta o método responsável por ajustar os pesos das sinapses dos neurônios, onde são percorridas todas as camadas após a primeira. Percebe-se que os atributos dos neurônios, valor de entrada (`x`), erro (`e`), impulso anterior (`dw`), assim como algumas constantes, `m_dEta` e `m_dAlpha` são utilizados para definir o novo peso da sinapse (linha 21).

Quadro 19 - Código fonte treinamento RNA - ajuste pesos sinapses

```
1  void CBackPropagation::AjustaPesoSinapses()
2  {
3      int i = 0;
4      int j = 0;
5      int k = 0;
6
7      double x = 0.0;
8      double e = 0.0;
9      double dw = 0.0;
10
11     for (i = 1; i < m_nNumCamadas; i++)
12     {
13         for (j = 0; j < m_pCamadas[i].m_nNumNeuronios; j++)
14         {
15             for (k = 0; k < m_pCamadas[i - 1].m_nNumNeuronios; k++)
16             {
17                 x = m_pCamadas[i - 1].m_pNeuronios[k].x;
18                 e = m_pCamadas[i].m_pNeuronios[j].e;
19                 dw = m_pCamadas[i].m_pNeuronios[j].dw[k];
20
21                 m_pCamadas[i].m_pNeuronios[j].w[k] += m_dEta * x * e + m_dAlpha * dw;
22                 m_pCamadas[i].m_pNeuronios[j].dw[k] = m_dEta * x * e;
23             }
24         }
25     }
26 }
```

O Quadro 20 demonstra o código responsável por ajustar o valor do erro na camada de saída. Os erros são ajustados com base no valor de entrada do neurônio (`x`) e na diferença entre a entrada e o valor desejado para a saída (linha15). Por fim, calcula-se a média de erro para o sinal (`m_dMAE`, linhas 17 e 20).

Quadro 20 - Código fonte treinamento RNA - erros saída

```
1  void CBackPropagation::ComputaErroSaida(double* saidaDesejada)
2  {
3      int i = 0;
4
5      double x = 0.0;
6      double d = 0.0;
7
8      m_dMAE = 0.0;
9
10     for (i = 0; i < m_pCamadas[m_nNumCamadas - 1].m_nNumNeuronios; i++)
11     {
12         x = m_pCamadas[m_nNumCamadas - 1].m_pNeuronios[i].x;
13         d = saidaDesejada[i] - x;
14
15         m_pCamadas[m_nNumCamadas - 1].m_pNeuronios[i].e = m_dGain * x * (1.0 - x) * d;
16
17         m_dMAE += fabs(d);
18     }
19
20     m_dMAE /= m_pCamadas[m_nNumCamadas - 1].m_nNumNeuronios;
21 }
```

O Quadro 21 demonstra como o sinal de entrada é propagado. Nota-se que a partir da primeira camada (linha 6), para cada neurônio realiza-se a soma ponderada da saída de todos os neurônios da camada imediatamente anterior (linhas 12, 13 e 14).

Quadro 21 - Código fonte treinamento RNA - propagação do sinal

```
1  void CBackPropagation::PropagaSinalRede()
2  {
3      int i = 0, j = 0, k = 0;
4      double sum = 0.0, saida = 0.0, w = 0.0;
5
6      for (i = 1; i < m_nNumCamadas; i++)
7      {
8          for (j = 0; j < m_pCamadas[i].m_nNumNeuronios; j++)
9          {
10             sum = 0.0;
11
12             for (k = 0; k < m_pCamadas[i - 1].m_nNumNeuronios; k++)
13             {
14                 saida = m_pCamadas[i - 1].m_pNeuronios[k].x;
15                 w = m_pCamadas[i].m_pNeuronios[j].w[k];
16                 // soma ponderada multiplicados pelo peso da sinapse
17                 sum += w * saida;
18             }
19
20             //sigmóide
21             m_pCamadas[i].m_pNeuronios[j].x = 1.0 / (1.0 + exp(-m_dGain * sum));
22         }
23     }
24 }
```

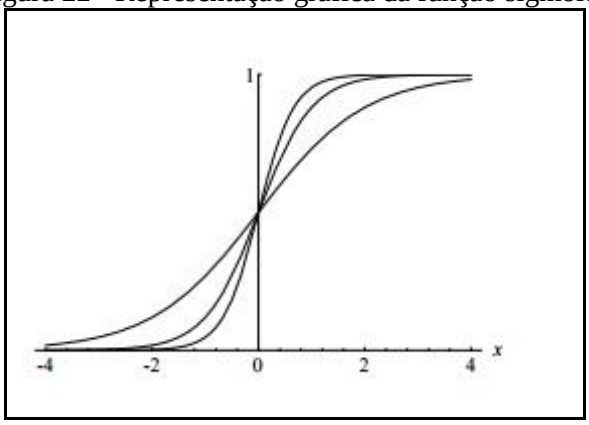
O valor acumulado pela soma ponderada é utilizado para calcular o valor da saída do neurônio atual (linha 21). O cálculo é realizado pela função de ativação sigmoide, que é responsável por gerar a saída do neurônio, a decisão. Como a função não é linear, torna a saída do neurônio uma transição contínua entre os níveis 0 e 1 (MARQUES, 2005). A Figura 21 representa matematicamente a função sigmoide e a Figura 22 representa o gráfico demonstrando a curvatura da função sigmóide.

Figura 21 - Fórmula sigmóide

$$\frac{1}{1+\exp\left(\sum_{i=1}^{n} w_i x_i - \theta\right)}.$$

Fonte: Rojas (1996, p. 153).

Figura 22 - Representação gráfica da função sigmóide



Fonte: Rojas (1996, p. 152).

O Quadro 22 demonstra o código responsável por retropropagar o erro para os neurônios das camadas ocultas e de entrada. O algoritmo percorre as camadas de trás para frente desconsiderando a última camada conforme demonstrado na linha 6. Em cada camada são percorridos os neurônios (linha 8), para cada neurônio é realizada a soma do valor de saída dos neurônios da próxima camada multiplicado pelo valor do erro. O valor é utilizado para calcular o novo valor do erro do neurônio atual (linha 16).

Quadro 22 - Código fonte treinamento RNA - retropropagando o erro

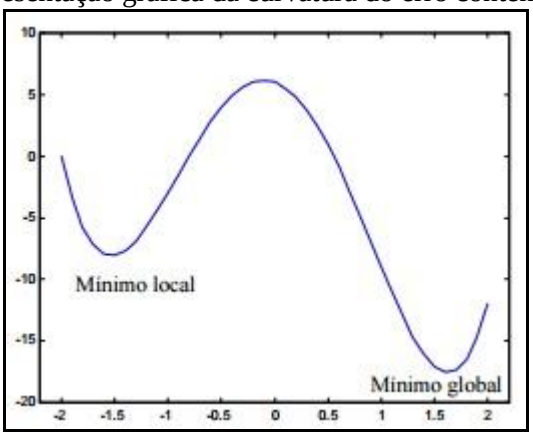
```
1  void CBackPropagation::RetropropagaErro()
2  {
3      int i = 0, j = 0, k = 0;
4      double x = 0.0, E = 0.0;
5
6      for (i = (m_nNumCamadas - 2); i >= 0; i--)
7      {
8          for (j = 0; j < m_pCamadas[i].m_nNumNeuronios; j++)
9          {
10             E = 0.0;
11
12             for (k = 0; k < m_pCamadas[i + 1].m_nNumNeuronios; k++)
13                 E += m_pCamadas[i + 1].m_pNeuronios[k].w[j] * m_pCamadas[i + 1].m_pNeuronios[k].e;
14
15             x = m_pCamadas[i].m_pNeuronios[j].x;
16             m_pCamadas[i].m_pNeuronios[j].e = m_dGain * x * (1.0 - x) * E;
17         }
18     }
19 }
```

3.3.5 Treinamento da RNA

Após a implementação da RNA, o foco foi alterado para estimar quantas iterações do algoritmo *backpropagation* seriam necessárias para atingir o objetivo de classificar os padrões para reconhecimento de sinais. Nesta etapa foram enfrentados alguns problemas característicos do algoritmo, a RNA simplesmente não conseguia classificar os padrões, o

problema ocorria devido ao chamado mínimo local. Na implementação clássica do *backpropagation* a RNA ajusta os pesos sinápticos dos neurônios a medida que a média de erro diminui. Porém, quando a média de erro aumenta são restaurados os pesos da iteração anterior e o treinamento é abortado. Para solucionar o problema essa tratativa foi desconsiderada. O algoritmo foi ajustado para salvar os pesos sinápticos somente quando a média de erro da RNA fosse inferior ao mínimo já definido. Dessa forma, não era necessário interromper o tratamento se a média de erro subisse, situação dos mínimos locais, bastava não considerar tais valores e continuar o treinamento. A Figura 23 traz a representação gráfica da curvatura de erro do mínimo local e global.

Figura 23 - Representação gráfica da curvatura do erro contendo mínimo local



Fonte: Zuben e Attux (2015, p. 53).

Por fim, a definição da quantidade de iterações depende diretamente do conjunto de entradas apresentado a RNA. Desta forma, optou-se por tornar o parâmetro flexível, o mesmo foi adicionado à tela de treinamento para que o usuário defina a quantidade que melhor satisfaz sua necessidade.

3.3.6 Reconhecimento de sinais

O Quadro 23 demonstra o código fonte responsável pela classificação da amostra de entrada. O método `ClassificarAmostra` recebe como parâmetro uma instância do tipo `CAmostraSinal` previamente tratada conforme mencionado na seção 3.3.2.

Quadro 23 - Classificar amostra de sinal

```
1  list<CConjuntoSaida> CBackPropagation::ClassificarAmostra(CAmostraSinal oAmostraSinal)
2  {
3      list<CConjuntoSaida> listSaidaRede;
4      CConjuntoSaida oSaidaRede;
5
6      list<CConjuntoCoordenadas> listConjuntoCoordenadas = oAmostraSinal.GetConjuntoCoordenadas();
7      list<CConjuntoCoordenadas>::iterator itConjuntoCoordenadas = listConjuntoCoordenadas.begin();
8      list<CConjuntoCoordenadas>::iterator itConjuntoCoordenadasFinal = listConjuntoCoordenadas.end();
9
10     for (; itConjuntoCoordenadas != itConjuntoCoordenadasFinal; itConjuntoCoordenadas++)
11     {
12         SetSinalEntrada(itConjuntoCoordenadas->m_dCoordenadas);
13
14         PropagaSinalRede();
15
16         GetSinalSaida(oSaidaRede.m_dSaida);
17
18         listSaidaRede.push_back(oSaidaRede);
19     }
20
21     return listSaidaRede;
22 }
```

Semelhante ao processo de treinamento, para cada conjunto de amostra contido no sinal, primeiro é informado o sinal de entrada (linha 12), posteriormente propaga-se o sinal pela RNA (já devidamente treinada) e recupera-se o sinal de saída (linhas 14 e 16 respectivamente). O conjunto de sinais de saída classificados será adicionado a uma lista de conjuntos de saída. A partir dos elementos da lista retornada será extraído o código relativo ao sinal, o código será consultado no cadastro e caso haja sinal correspondente o mesmo será considerado

3.3.7 Representação gráfica dos sinais

Neste ponto vale mencionar o componente `WebBrowser` disponibilizado na IDE do Visual Studio. Tal como remete o nome, o componente é um `web browser`. O componente foi muito útil para representar graficamente a amostra selecionada. Isso porque para desenhar o objeto 3D relativo à amostra foi utilizado a biblioteca LeapJS-PlayBack. A biblioteca foi disponibilizada na internet pela empresa Leap Motion, a mesma foi implementada em JavaScript e pode ser utilizada para gravar, desenhar o objeto 3D e salvar dos dados dos frames do dispositivo Leap Motion. Neste caso a biblioteca foi adaptada para rodar sobre o *internet information service* da Microsoft. Desta forma, foi possível vincular o endereço web local da mesma para representar graficamente as amostras.

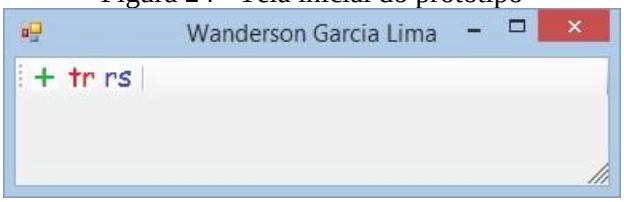
O componente ainda exigiu algumas alterações no registro do Windows para que fosse utilizada a última versão do *Internet Explorer* para interpretar a página *web*. Isso porque o componente `web browser` usa por padrão as configurações do *Internet Explorer* versão 6. Os ajustes no registro do Windows foram necessários porque a página `web` que utiliza a biblioteca LeapJS-PlayBack obrigatoriamente deve utilizar o marcador `canvas`. Desta forma, é

necessário que a última versão do *Internet Explorer* instalada na máquina que for executar o protótipo seja compatível com o HTML5.

3.3.8 Operacionalidade da implementação

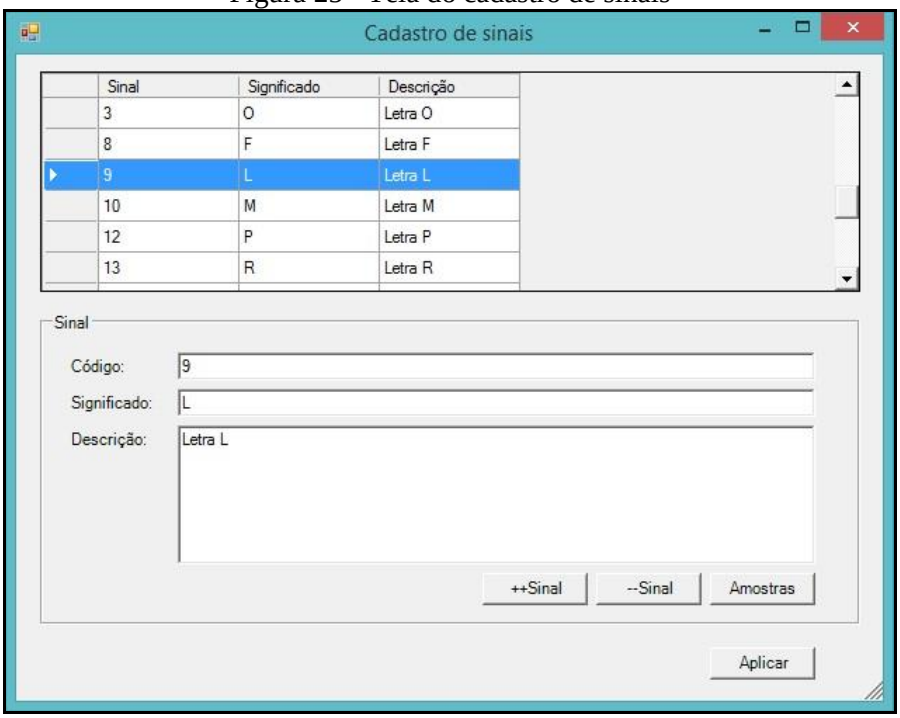
Nesta seção são apresentadas as funcionalidades destacadas nos casos de uso contidos na seção 3.2.1, começando pela Figura 24 que ilustra a tela inicial do protótipo onde são acionadas as funcionalidades através de ícones na `statusbar`.

Figura 24 - Tela inicial do protótipo



A Figura 25 demonstra a tela da funcionalidade de cadastro de sinais. Nesta tela, o usuário tem a possibilidade de incluir sinais, através do botão `++Sinal`, excluir sinais através do botão `--Sinal` ou selecionando o sinal na grade e pressionando a tecla `delete` e, vincular amostras ao sinal através do botão `Amostras`. Observa-se que nela ainda consta o botão `Aplicar`. O mesmo é utilizado para persistir as alterações realizadas, caso o mesmo não seja acionado ao sair da tela, todas as alterações são descartadas.

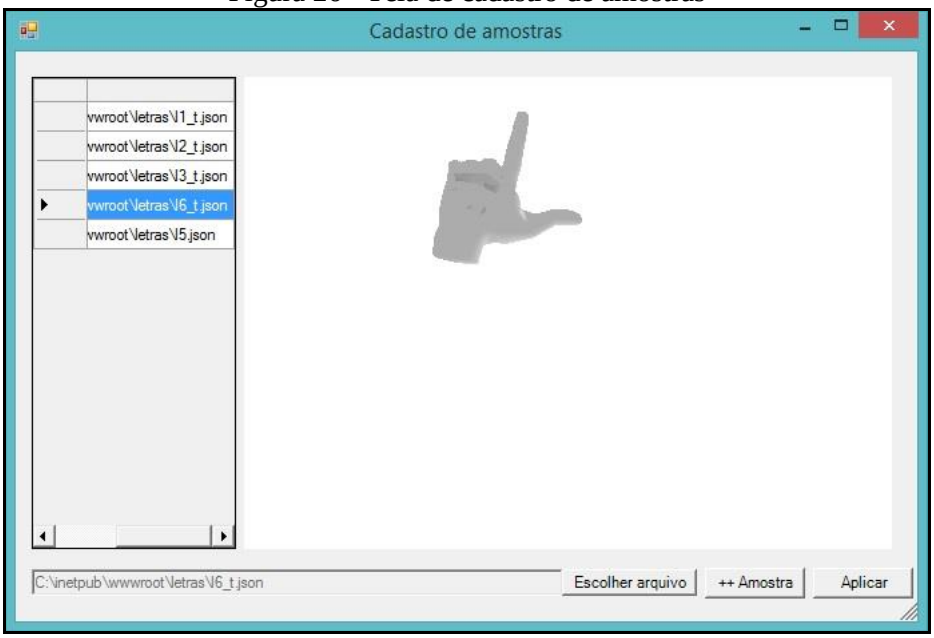
Figura 25 - Tela do cadastro de sinais



A Figura 26 apresenta a tela da funcionalidade cadastro de amostras. Na ilustração são apresentadas as amostras referentes ao sinal da letra `L`. Nota-se no lado esquerdo, que foram

adicionadas cinco amostras. O processo de adição inicializa-se selecionando o arquivo no formato `JSON` exportado pela aplicação LeapJS Playback. Pode-se selecionar o arquivo após clicar no botão `Escolher Amostra`. Na sequência a amostra é incluída na lista através do botão `++Amostra`. Para excluir amostras da lista, deve-se selecionar a mesma e pressionar a tecla `delete`. Ao final, utiliza-se o botão `Aplicar` para efetuar as alterações.

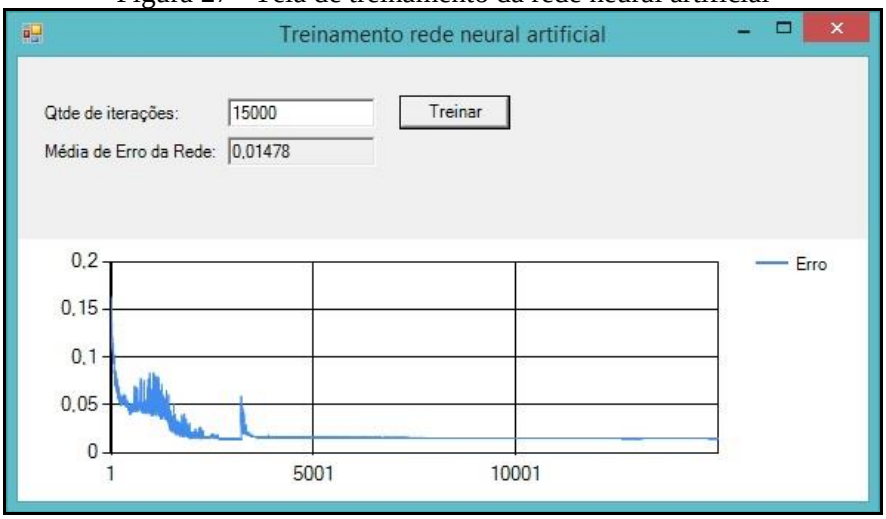
Figura 26 - Tela de cadastro de amostras



A Figura 27 apresenta a funcionalidade de treinamento da RNA. Esta tela possui apenas dois componentes para interação com usuário, o `editbox Qtde de iteração` e o botão `Treinar`. A quantidade de iterações, como se pode imaginar, é utilizada para definir a quantidade de vezes que o algoritmo *backpropagation* irá ser executado no processo de treinamento. Ao clicar neste botão, inicia-se o processo de treinamento.

Ao inicializar o treinamento o título da tela será alterado para "*Treinando – Aguarde...*" e o mesmo voltará à expressão original ao final do processo. Também ao final do processo, é exibido e/ou atualizado o gráfico de linha relativo a média de erros no decorrer das iterações definidas pelo usuário. O gráfico é meramente ilustrativo e visa fornecer um *feedback* mínimo ao usuário em relação a comportamento da RNA durante o treinamento da mesma.

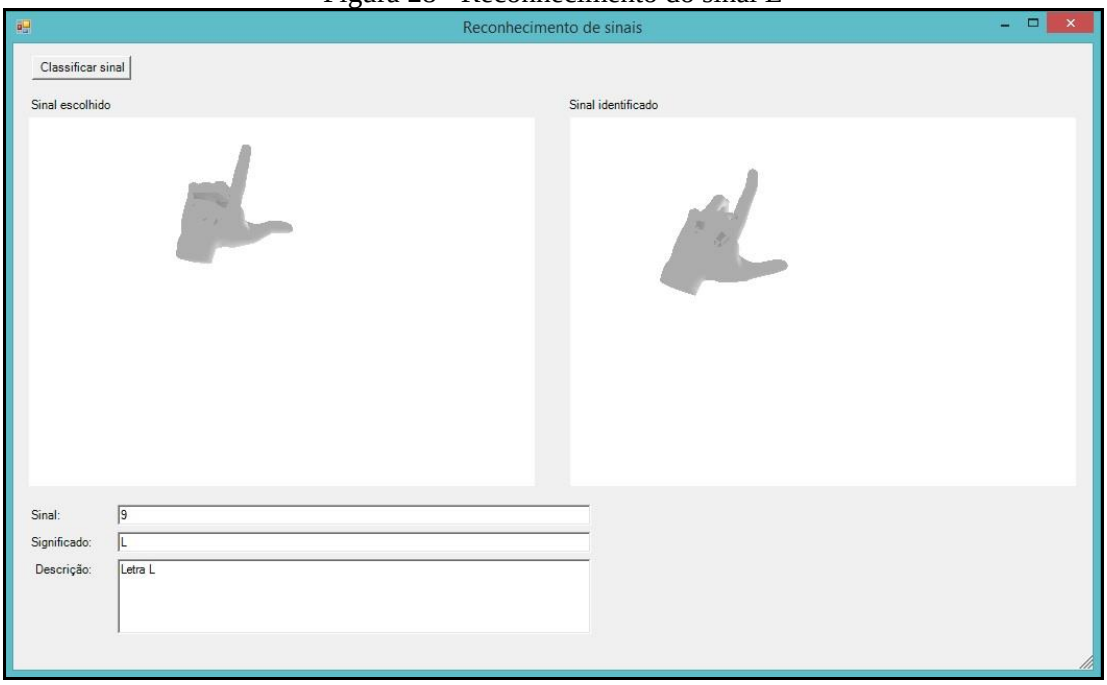
Figura 27 - Tela de treinamento da rede neural artificial



A funcionalidade de reconhecimento de sinais é apresentada na Figura 28. Essa funcionalidade conta com o botão `Classificar Sinal`, localizado no canto superior esquerdo da tela. Ao ser clicado, ele exibirá a tela de seleção de arquivos, padrão do Windows, onde o usuário poderá escolher uma amostra de sinal para ser reconhecida. Após selecionar a amostra, o protótipo irá classificá-la e buscar pelo sinal compatível. Três situações são possíveis neste momento:

a) sinal é reconhecido e tem sua representação gráfica expressa na tela;
b) sinal não é reconhecido, porém a classificação identificou saída compatível com sinal existente no cadastro. Neste caso, é exibida a mensagem “o erro extrapola o limite máximo definido”;
c) o sinal não é reconhecido e o classificador não identifica saída compatível com nenhuma amostra presente no cadastro.

Figura 28 - Reconhecimento do sinal L



3.4 RESULTADOS E DISCUSSÕES

Esta seção tem como objetivo demonstrar a metodologia de testes utilizada e os resultados obtidos na fase de validação do protótipo. Na seção 3.4.1 são definidos os grupos de amostras de testes e na seção 3.4.2 são demonstrados os experimentos realizados para estabelecer a taxa de reconhecimento do protótipo desenvolvido.

3.4.1 Amostras de testes

Após a realização de diversos testes, treinando a rede e realizando o reconhecimento de sinais, percebeu-se que o sucesso de uma RNA está diretamente relacionado ao conjunto de entrada. Neste ponto, decidiu-se separar os sinais em três grupos distintos. O primeiro grupo foi composto por todas as letras do alfabeto datilológico. O segundo grupo foi composto por todas as letras correspondentes a todos os sinais contidos no alfabeto datilológico cujo sinal não possui o parâmetro de movimento, denominados sinais estáticos. O terceiro grupo foi composto pelos sinais com parâmetro de movimento, denominados sinais dinâmicos.

3.4.2 Reconhecimento de sinais

Esta seção aborda os experimentos realizados para validar a eficiência da RNA ao reconhecer os sinais. A mesma foi separada em três seções. A seção 3.4.2.1 apresenta os resultados dos testes do conjunto de treinamento contendo os sinais estáticos e dinâmicos. A

seção 3.4.2.2 exibe os resultados considerando o conjunto de treinamento com sinais estáticos. Por fim, a seção 3.4.2.3 apresenta os resultados do conjunto de sinais dinâmicos. Vale ainda salientar que nos três casos a quantidade de iterações no treinamento foi definida em 4.000 iterações.

3.4.2.1 Experimento 01: reconhecimento de sinais estáticos e dinâmicos

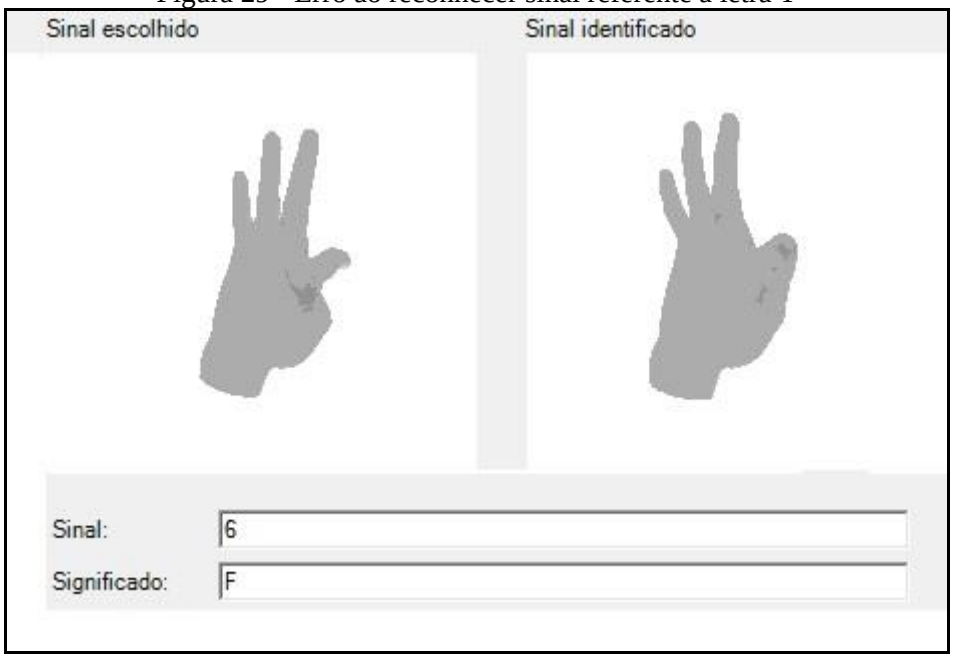
Neste experimento, a RNA foi treinada para reconhecer o grupo de amostras contendo sinais estáticos e dinâmicos. Os resultados podem ser vistos na Tabela 1. O tempo de treinamento da RNA para este conjunto de entrada foi de aproximadamente 4 horas executando em uma máquina com processador Intel® Core™ i5 1.6 GHz contendo 4GB de memória DDR3.

Tabela 1 - Tabela de resultados do reconhecimento de sinais estáticos e dinâmicos

Amostra	Quantidade de amostras	Quantidade de acerto	Taxa acerto (%)
A	2	1	50,00
B	2	2	100,00
C	3	2	66,67
D	3	2	66,67
E	4	0	0,00
F	8	8	100,00
G	5	0	0,00
H	3	3	100,00
I	4	2	50,00
J	3	3	100,00
K	7	7	100,00
L	7	6	85,71
M	6	6	100,00
N	3	1	33,33
O	7	3	42,86
P	10	7	70,00
Q	5	2	40,00
R	6	1	16,67
S	4	1	25,00
T	12	5	41,67
U	3	1	33,33
V	9	8	88,89
W	3	3	100,00
X	5	4	80,00
Y	6	6	100,00
Z	4	4	100,00
		Média da rede	**65,03**

Os resultados expressos na Tabela 1 mostram que alguns sinais tiveram a taxa de acerto baixa. Vale destacar os sinais das letras E, G, O, N, R, S, T e U. Os sinais E e S apresentaram conflito no treinamento pela semelhança entre si, o mesmo ocorre com os sinais G e D, T e F. As letras N, O e U tiveram uma taxa de acerto baixa porque a qualidade dos sinais extraídos do Leap Motion não foram boas. Não por deficiência do aparelho, mas sim por falhas no momento ao escolher as melhores amostras. A letra R não evidenciou claramente o motivo da baixa taxa de acerto, sendo que para essa letra, considerou-se que a rede não foi capaz de classificar o padrão. Juntos os sinais mencionados fazem com que a média de acerto da RNA fique em torno de 65%. Enquanto que a média da taxa de acerto fica em torno 83% se desconsiderados tais sinais. A Figura 29 demonstra alguns dos problemas no reconhecimento de sinais. Devido à semelhança, algumas das amostras da letra T são classificadas como F.

Figura 29 - Erro ao reconhecer sinal referente à letra T



A Figura 29 demonstra o resultado do protótipo ao tentar realizar o reconhecimento de uma amostra da letra T `(Sinal escolhido)`. Ao classificar a amostra a RNA teve com resultado saída compatível com a letra F `(Sinal identificado)`. Esse problema ocorre porque as amostras possuem características comuns entre si, os dedos mínimo, anelar e médio esticados e pouco afastados entre si e o dedo indicador apontando para frente. A característica que difere as amostras é o posicionamento do dedo polegar, na letra T sob o indicador, na letra F sobre o indicador, devido à pequena variação entre as letras, a RNA acabou classificando incorretamente algumas amostras.

3.4.2.2 Experimento 02: reconhecimento de sinais estáticos

Neste experimento, a RNA foi treinada para reconhecer o grupo de amostras contendo apenas sinais estáticos. Os resultados podem ser vistos na Tabela 2.

Tabela 2 - Tabela de resultados do reconhecimento de sinais estáticos

Amostra	Quantidade de amostras	Quantidade de acerto	Taxa acerto (%)
A	2	2	100,00
B	2	2	100,00
C	2	2	100,00
D	3	3	100,00
E	4	0	0,00
F	8	6	75,00
G	5	1	20,00
I	4	2	50,00
L	6	5	83,33
M	6	5	83,33
N	3	1	33,33
O	7	3	42,86
P	10	7	70,00
Q	5	3	60,00
R	6	4	66,67
S	4	2	50,00
T	12	7	58,33
U	3	3	100,00
V	9	9	100,00
W	3	3	100,00
X	5	4	80,00
		Média da rede	**70,14**

Nos resultados representados na Tabela 2, percebe-se que os sinais das letras E, G, N, O e S fazem com que a média de acerto da RNA seja novamente baixa. Porém, os sinais R e U tiveram uma melhor representatividade, fato que aumentou um pouco a taxa de acerto para aproximadamente 70%. O tempo de treinamento da RNA foi de aproximadamente 3 horas utilizando a mesma máquina mencionada na seção 3.4.2.1.

3.4.2.3 Experimento 03: reconhecimento de sinais dinâmicos

Neste experimento, foram adicionadas amostras das palavras `chuva`, `nome`, `oi`, `queimar` e `Wanderson` ao grupo de sinais dinâmicos, cujo objetivo foi aumentar o grupo de teste para verificar a eficiência da RNA em classificar sinais compostos por ambas as mãos, no caso, os sinais `chuva` e `queimar`. Os resultados são demonstrados na Tabela 3.

Tabela 3 - Tabela de resultados do reconhecimento de sinais dinâmicos

Amostra	Quantidade de amostras	Quantidade de acerto	Taxa acerto (%)
Chuva	5	5	100,00
H	3	3	100,00
J	3	3	100,00
K	7	7	100,00
Nome	5	5	100,00
Oi	6	6	100,00
Queimar	6	6	100,00
Wanderson	3	3	100,00
Y	3	3	100,00
Z	6	5	83,33
		Média da rede	**98,33**

A partir dos resultados da Tabela 3, pode-se perceber que a taxa de acerto foi de aproximadamente 98%. Vale ressaltar que apesar do grupo conter menos amostras que os demais grupos de treinamento, a alta taxa de acerto já foi verificada nos testes da seção 3.4.2.1 onde todos os sinais dinâmicos tiveram taxa de acerto de 100%. O teste ainda comprova a capacidade do protótipo de classificar sinais que utilizam ambas as mãos.

4 CONCLUSÕES

Tendo em vista o objetivo de reconhecer sinais em LIBRAS sensoriados pelo dispositivo Leap Motion utilizando RNA, conclui-se que a utilização de RNAs através do algoritmo *backpropgation* é satisfatória para realizar a classificação dos padrões de entrada permitindo dessa forma realizar o reconhecimento dos sinais. Os testes realizados demonstram a capacidade do protótipo em reconhecer diversos sinais (tanto estáticos como dinâmicos), atingindo dessa forma seu propósito. Outro fato perceptível nos testes foi a diferença nos resultados em relação aos grupos de treinamento, tal fato reforçou a premissa que a quantidade de elementos no conjunto de entrada e o tipo do problema impactam diretamente no resultado da RNA, visto que a mesma apresentou resultados melhores quando possuía menos dados para classificar.

As tecnologias utilizadas atenderam as demandas e o dispositivo Leap Motion se mostrou eficaz para realizar o sensoriamento dos sinais. O mesmo possibilitou o reconhecimento de sinais com movimentos, uma demanda não atendida pelos trabalhos correlatos.

4.1 EXTENSÕES

Dentre as extensões possíveis destaca-se a captura dos sinais sensoriados diretamente do dispositivo Leap Motion. Isto iria eliminar a dependência da aplicação LeapJS Playback. Em relação à RNA, pode-se explorar a otimização do treinamento nas funções de ativação e definição dos erros. Isto, possivelmente sanaria as limitações ao reconhecer sinais semelhantes, como no caso dos sinais F e T. Outro ponto de melhoria seria disponibilizar uma funcionalidade para permitir alterar a topologia da RNA. Permitir definir a topologia em tempo de execução, tornaria o protótipo mais flexível, pois seria possível ajustá-lo conforme a demanda do usuário.

REFERÊNCIAS

ACESSIBILIDADE BRASIL. [S.l.], 2015. Disponível em: <http://www.acessibilidadebrasil.org.br/versao_anterior/>. Acesso em: 10 jun. 2015.

AMO FAZER ATIVIDADES. [S.l.], 2015. Disponível em: <http://amofazeratividades.blogspot.com.br>. Acesso em : 10 jun. 2015.

BAMBINETI , Rodrigo. **Ferramenta de reconhecimento de gestos da mão.** 2009. 60 f. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

BARTHELEMY, Sylvain. **Free C++ Neural Network Source Code.** [S.l.], 2000. Disponível em <http://www.sylbarth.com/mlp.php>. Acesso em 29 maio 2015.

BRASIL. Lei n° 10.436, de 24 de abril de 2002. Dispõe sobre a língua Brasileira de Sinais - Libras e dá outras providências. **Diário Oficial da União**. Brasília: Disponível em: <https://www.presidencia.gov.br/ccivil_03/Leis/2002/L10436.htm>. Acesso em: 17 jun. 2015.

GESSER, Audrei. **Libras? Que língua é essa?**: Crenças e preconceitos em torno da língua de sinais e da realidade surda. São Paulo: Parábola Editorial, 2009.

KINECT.[S.l.], 2015. Disponível em <http://www.microsoft.com/en-us/kinectforwindows/>. Acesso em: 10 jun. 2015.

LEAP MOTION. [S.l.], 2015. <https://developer.leapmotion.com/getting-started>. Acesso em: 17 jun. 2015.

MARQUES, Jorge Salvador. **Reconhecimento de padrões:** Métodos estatísticos e neurais. Lisboa: IST Press, 2005.

MARTINS, Sílvia Adosinda d'Assunção. **As dificuldades de comunicação entre surdos e ouvintes:** Propostas de soluções. 2012. 105 f. Dissertação (Mestre em Ciências da Educação) - Escola Superior de Educação Almeida Garrett, Lisboa.

NINTENDO. [S.l.], 2015. Disponível em: <http://www.nintendo.com/> Acesso em: 10 jun. 2015.

POTTER, Leigh; ARAULLO, Jake; CARTER, Lewis. **The Leap Motion controller:** A view on sign language. 2013. Griffith University, Nathan - Austrália.

ROJAS, R; **Neural networks.** 1996. Springer-Verlag, Berlim.

SANTANA, Ana Paula. **Surdez e Linguagem** - aspectos e implicações neurolinguísticas. São Paulo: Plexus, 2007.

SANTIN, Diego M. **Ferramenta para transcrição do alfabeto datilológico para texto utilizando Microsoft Kinect.** 2013. 59 f. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

SECRETARIA DE EDUCAÇÃO ESPECIAL. **O tradutor e intérprete de língua brasileira de sinais e língua portuguesa:** programa nacional de apoio à educação de surdos. 2004. 94 f. il. Brasília: MEC; SEESP. Disponível em: <http://portal.mec.gov.br/seesp/arquivos/pdf/tradutorlibras.pdf>. Acesso em 26 jun. 2015.

STROBEL, Karin Lilian; FERNANDES, Sueli. **Aspectos linguísticos da LIBRAS:** língua brasileira de sinais. 2008. Secretaria de Estado da Educação, Superintendência de Educação. Departamento de Educação Especial. Curitiba: SEED/SUED/DEE. Disponível em: <http://www.librasgerais.com.br/materiais-inclusivos/downloads/Aspectos-linguisticos-da-LIBRAS.pdf>. Acesso em 16 jun. 2015.

TAFNER, Malcon A; XEREZ, Marcos de; RODRIGUES FILHO, Ilson W. **Redes neurais artificiais**: introdução e princípios da neurocomputação. Blumenau: Eko, 1996.

TAVARES, Ilda Maria Santos; CARVALHO, Tereza Simone Santos de. **Educação inclusiva e práticas educativas:** O ensino de libras para ouvintes. V Fórum Identidades e Alteridades I Congresso Nacional Educação e Diversidade (2011). UFS Itabaiana.

TISSOT, Hegler C; CAMARGO, Luiz C; POZO, Aurora T. R. **Treinamento de redes neurais feedforward**: comparativo dos algoritmos backpropagation e differential evolution. 2012. 12 f. Departamento de Informática, Universidade Federal do Paraná, Curitiba. Disponível em: <http://www.lbd.dcc.ufmg.br/colecoes/enia/2012/0020.pdf>. Acesso em: 12 jun. 2015.

ZUBEN, Fernando J V; ATTUX, Romis R F. **Perceptron de Múltiplas Camadas**. [S.l.], 2015. Disponível em: <ftp://ftp.dca.fee.unicamp.br/pub/docs/vonzuben/ia353_1s07/topico5_07.pdf>. Acesso em: 12 jun. 2015.