Centro Universitário de Brasília

Faculdade de Tecnologia e Ciência Sociais Aplicadas – FATECS

Engenharia de Computação

Alessandra de Matos Frossard

MECANORRECEPTORES ARTIFICIAIS APLICADOS A PRÓTESES DE MEMBRO SUPERIOR

Brasília
2020

Centro Universitário de Brasília
Faculdade de Tecnologia e Ciência Sociais Aplicadas – FATECS

Engenharia de Computação

Alessandra de Matos Frossard

MECANORRECEPTORES ARTIFICIAIS APLICADOS A PRÓTESES DE MEMBRO SUPERIOR

Produto apresentado para a conclusão do 18º Programa de Iniciação Científica do Centro Universitário de Brasília – UniCEUB, orientado pelo professor Francisco Javier de Obaldía Díaz

Brasília
2020

RESUMO

Atualmente, a taxa de rejeição de próteses de membros superiores pode chegar até 70%. Isso se dá por diversos fatores, tanto físicos quanto psicológicos. Um dele é a inexistência de uma via bilateral prótese-objeto que gere feedbacks sensoriais, o que ajudaria no reconhecimento de um objeto e na fluidez dos movimentos. Este projeto tem como objetivo melhorar essa adversidade através da criação de uma rede neural (RNA) *Feedforward* Perceptron Multicamadas com *Backpropagation*, que, tendo como entrada a posição da mão ao segurar um objeto, reconheça-o e indique a preensão apropriada. Para isso criou-se uma simulação de uma mão 3D no Blender 2.9 que gera posições aleatórias dos dedos da mão, criando um banco de dados que alimenta o a rede neural criada para no seu treinamento, objetivando chegar nos pesos sinápticos mais apropriados. Assim, ao colocar-se uma entrada qualquer, a rede informará qual é a preensão mais próxima. A RNA criada utiliza como função de ativação a função Sigmoid, tem cinco neurônios na camada de entrada, quatro na camada oculta e três na camada de saída. Foi criado um banco de dados de 40 entradas, e ao utilizá-lo na fase de treinamento da rede Perceptron, durante 10000 épocas, chegou-se a duas matrizes de pesos, uma para a camada oculta e uma para a camada, que tem como percentagem de erro 2%. Pode-se concluir que redes neurais são uma boa solução para esse problema, pois mesmo não sendo a melhor rede neural possível de ser construída, ela já mostrou um grande avanço no reconhecimento de objetos, e assim, pode ser melhorada de diversas formas para ser utilizada em próteses reais, melhorando a sua aceitação.

Palavras-Chave: Rede Neural. Perceptron Multicamadas. Próteses de Membros Superiores. Banco de Dados

1. Introdução

Próteses de membro superior, apesar de terem como objetivo trazer uma maior independência a seus usuários e maior autoestima, são altamente rejeitadas, podendo chegar a ter uma taxa de abandono de 70% segundo dados obtidos na AACD (Associação de Assistência à Criança Defeituosa) (Carvalho, 2004). Uma possível razão para o baixo índice de aceitação decorre da complexidade da mão e da quantidade de movimentos diários que dependem dela. A falta de feedback sensorial das próteses atuais limita sua utilização e torna robótico o movimento.

As próteses e suas adversidades são assuntos multidisciplinares, pois agregam não somente esforços de engenharia na sua projeção, criação e execução, mas também nas áreas de medicina, ao tratar-se de amputações, psicologia, no cuidado mental e emocional do indivíduo no processo de aceitação de sua situação, fisioterapia e terapia ocupacional e as demais áreas relacionadas com a reabilitação do paciente.

Esse projeto propõe melhorar a movimentação da prósteses através da criação de uma rede neural Perceptron Multicamada (MLP) em Python, utilizada em uma simulação de um protótipo de mão no Blender 2.9, que classifica a preensão utilizada de acordo com as coordenadas espaciais dos dedos que identifica a posição dos objetos. Ao final, o sistema inteligente é capaz de decidir o movimento adequado para segurar o objeto em questão.

A principal motivação desse trabalho foi tentar unir os conhecimentos de Engenharia adquiridos e utilizá-los na melhoria de um problema social, tentando responder às perguntas:

- É possível criar uma via bilateral prótese-objeto de troca de informações que melhore a aceitação de tal produto?

- Como que a inteligência artificial pode ajudar a reduzir problemas sociais?

1.1 Objetivos

1.1.1 Objetivo Geral

Inicialmente, o projeto tinha como objetivo criar um sistema de circuito sensorial utilizando sensores de toque e pressão conectados a um micro controlador PIC programado com um software de rede neural artificial Perceptron implementado computacionalmente para receber as informações coletadas e assim identificar a posição dos objetos. Assim, o

estudo teria duas áreas de foco: software, na criação da RN, e hardware, na fabricação do circuito sensorial.

Contudo, devido aos imprevistos gerados pelo Covid-19, teve-se que adaptar o projeto. Assim, o foco voltou-se somente para a parte de software na simulação do que seria a parte de hardware. Logo, o objetivo final do projeto foi criar uma rede neural Perceptron Multicamadas em Phyton, que é mais complexa que a planejada anteriormente, para a classificação de posições aleatórias dos dedos em três tipos de preensões diferentes, identificando, assim, no mínimo três objetos.

1.1.2 Objetivos Específicos

Para atingir o objetivo geral, os seguintes objetivos específicos foram identificados:

a) Desenvolveu-se uma simulação de prótese através da criação de um modelo de mão manipulável em 3D no Blender 2.9, que gera posições aleatórias dos dedos no espaço cartesiano dessa API (dentro dos padrões humanos possíveis.)
b) Estabeleceu-se uma base de dados de acordo com essas posições, para a alimentação da rede neural.
c) Criou-se uma rede neural Perceptron Multicamadas que utiliza os dados da base gerada para seu treinamento, gera os pesos adequados e classifica novas entradas nos três tipos de preensão.

2. Fundamentação Teórica

2.1 Preensão

A mão pode coordenar uma grande variedade de movimentos, e sua estrutura anatômica e funcional complexa converge principalmente para a realização das preensões (Nordin; Frankel, 2003, Kapandji, 2007). Preensão é a capacidade de utilizar a mão como uma pinça ou garra. Existem dois tipos básicos de preensão: a preensão palmar ou de força e a preensão de precisão ou pinça. A diferença entre os dois movimentos está na área de aplicação de força. A preensão de precisão é aplicada na ponta dos dedos, enquanto a de força, na palma da mão. (Costa, 2017). A escolha do movimento de preensão apropriado se faz através de um processo de percepção e decisão

A Figura 1, a seguir, mostra a taxonomia das preensões:

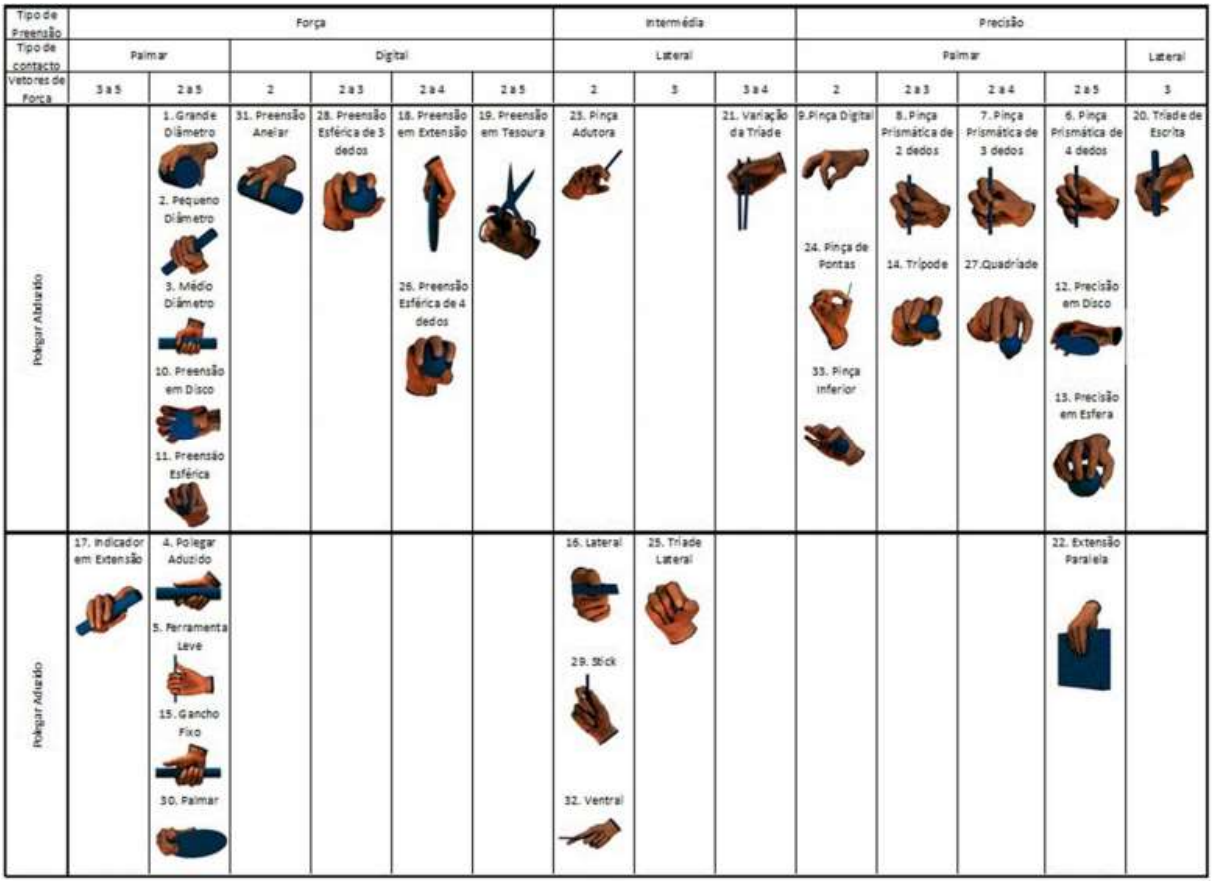


Figura 1. Taxonomia das Preensões. (Feix et al., 2016)

2.2 Redes Neurais

As Redes Neurais Artificiais (RNAs) equivalem à representação computacional simplificada de redes de neurônios biológicos. Elas têm sido aplicadas em tarefas de reconhecimento de padrões, memórias associativas, otimização de sistemas, controle de processos e de processamento de dados (da Silva et al, 2016). O uso destas redes oferece propriedades e capacidades úteis, tais como a não linearidade, adaptabilidade, tolerância a falhas e mapeamento entrada/saída (PUC-RIO)

A estrutura das redes neurais foi desenvolvida a partir dos modelos de sistemas nervosos biológicos. O modelo de neurônio mais simples foi proposto em 1943 por McCulloch & Pitts, e ainda é muito utilizado (da Silva et al, 2016). Ele está representado na Figura 2, a seguir:

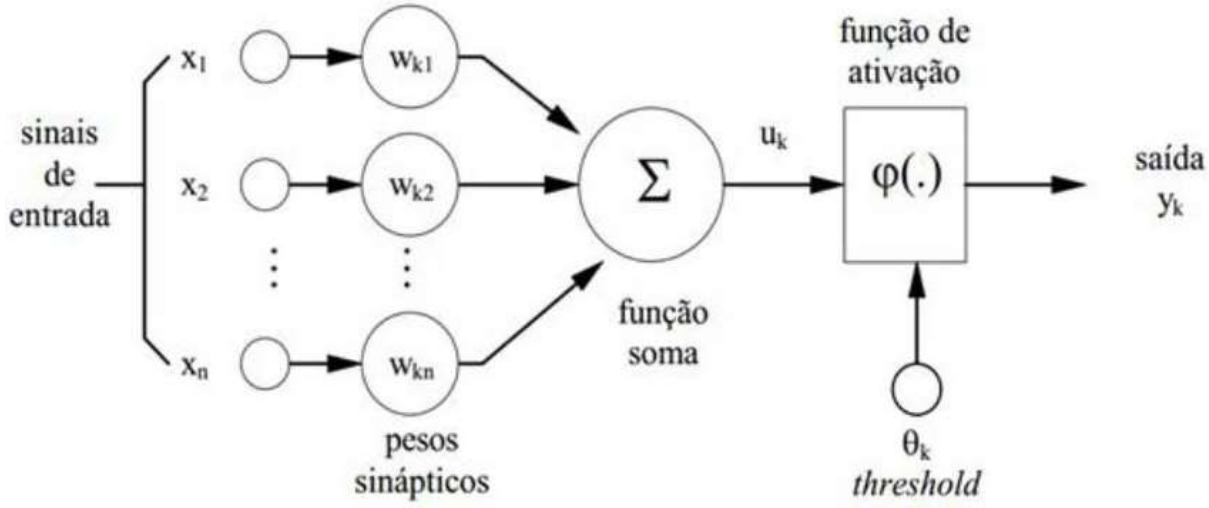


Figura 2. Neurônio Artificial

Pela imagem, observa-se sete elementos básicos: os sinais de entrada x_j, os pesos associados às sinapses w_{kj} (zonas ativas de contato entre uma terminação nervosa e outros neurônios), o limiar (bias) θ_k, o potencial de ativação u_k, a função de ativação φ(.), os sinais de saída y_k e o combinador linear da saída Σ.

Matematicamente, pode-se descrever o resultado produzido pelo neurônio artificial pelas duas expressões a seguir (da Silva et al, 2016):

$$u = \sum_{j=1}^{n} w_j * x_j - \theta \qquad (1)$$

$$y = \varphi(u) \qquad (2)$$

As RNAs se diferenciam entre si de acordo com número de camadas nas quais os neurônios são dispostos suas interligações e a forma que ocorre seu treinamento (com ou sem supervisão). O treinamento do RNA é a forma de alterar os pesos associados às sinapses, w_{kj}, de modo que ocorra a convergência para a representação de um modelo matemático, e assim o programa fornece uma saída condizente com as informações de entrada (PUC-RIO). O algoritmo desse treinamento é um conjunto de regras bem-definidas para a solução do aprendizado da rede neural e melhoria do seu desempenho.

É comum a existência de duas fases na prática de uma rede neural artificial: a fase de treinamento e a faze de validação. Na fase de treinamento, a rede ajusta seus pesos através do uso de um algoritmo apropriado. A saída da RNA é comparada com a saída desejada (alvo), e então o resíduo é usado para ajustar os pesos sinápticos de acordo com o algoritmo de treinamento. Na fase de validação, a RNA calcula a saída com base nas entradas e nos pesos (Costa, 2017).

2.2.1 Funções de Ativação

Tem como objetivo limitar a saída do neurônio dentro de um intervalo de valores razoáveis a serem assumidos pela própria imagem funcional. A função utilizada em cada rede neural depende do projeto que está sendo criado (da Silva et al, 2016). Alguns exemplos de funções de ativação são:

a) *Função Degrau* (Função de ativação parcialmente diferenciável)

$$f(x) = \begin{cases} 1, se\ u \geq 0 \\ 0, se\ u < 0 \end{cases} \qquad (3)$$

b) *Função Rampa Simétrica* (Função de ativação parcialmente diferenciável)

$$f(x) = \begin{cases} a, se\ u > a \\ u, se - a \leq u \leq a \\ -a, se\ u < -a \end{cases} \qquad (4)$$

c) *Função Logística/Sigmoid* (Função totalmente diferenciável)

$$f(x) = \frac{1}{1-e^{-x}} \qquad (5)$$

d) *Função Tangente Hiperbólica* (Função totalmente diferenciável)

$$f(x) = \frac{e^{2x}-1}{e^{2x}+1} \qquad (6)$$

2.2.2 Rede Perceptron Multicamadas (MLP)

As RNAs desse tipo são caracterizadas por terem uma ou mais camadas intermediárias (ocultas), entre a camada de entrada e a camada de saída, como pode ser visto na Figura 3. Ela pertence à arquitetura feedfoward de camadas múltiplas e seu treinamento é realizado de forma supervisionada e elas geralmente utilizam o algoritmo de aprendizagem denominado backpropagation, baseado na regra delta generalizada e na regra delta/competitiva.

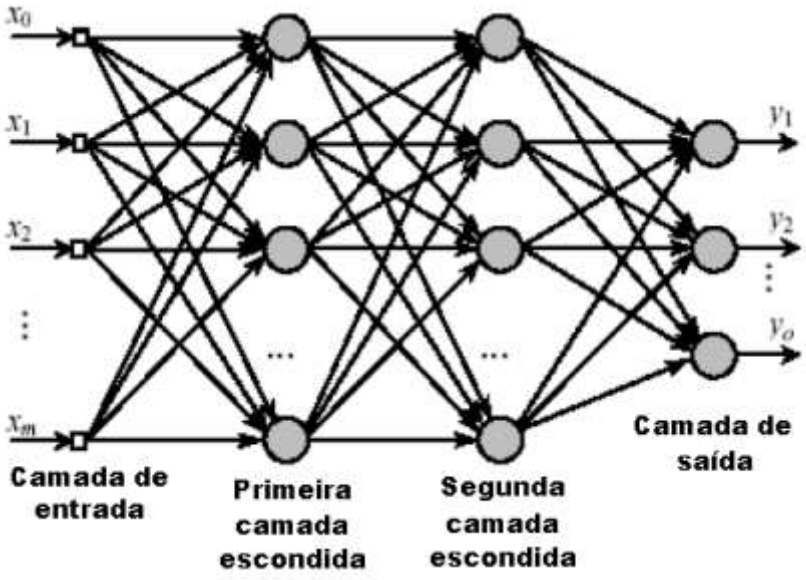


Figura 3. Exemplo de Rede do tipo MLP

3. Método

Primeiramente foi feita uma pesquisa tanto literária quanto de simuladores para adaptar a proposta inicial à nova situação. Estudou-se as opções, e a primeira ideia foi utilizar a biblioteca LibHand (http://www.libhand.org/) que permite a renderização e o reconhecimento das articulações da mão humana em 3D, seguindo trabalhos anteriores que foram realizados com ela (Szabó, 2020; Godoi, 2013). Ela pode ser implementada em C++ e oferece uma interface do MATLAB, tornando possível a análise dos dados por ali.

Assim, criou-se uma máquina virtual Ubuntu 20.04 (64 bits) através do Oracle VM VirtualBox, já que no site oficial da biblioteca, indica que funciona para MacOS e Ubuntu Linux, instalou-se o MATLAB, e a tentou-se instalar o LibHand e tudo que é pedido. O primeiro problema ocorreu ao ver-se que a memória do computador não era suficiente para tudo que estava sendo processado, tanto da máquina virtual, quanto da normal. Então utilizou-se um SSD para melhorar o processamento. O segundo problema ocorreu na instalação do LibHand, mais especificamente na biblioteca 'libtiff4-dev', e procurou-se uma biblioteca que o substituísse. Pois, mesmo depois de instalada, o computador não a reconhecia.

Dessa forma, desistiu-se da máquina virtual foi iniciada uma tentativa de instalar o LibHand no Windows, de acordo com as atualizações apresentadas no git da biblioteca (https://github.com/libhand/libhand). A instalação é bem mais complexa, com mais

programas para serem executados, e ocorreram erros em diversos momentos para os quais não encontrou-se uma solução. Nesse momento, desistiu-se da utilização do LibHand e buscou-se uma nova forma de criar uma simulação.

Uma outra solução iniciada foi a utilização do MATLAB Simulink Simscape Multibody, na tentativa de criar uma armadura de mão que pudesse ser utilizada no local de uma 3D. Porém essa solução também não funcionou, pois muito tempo estava sendo gastado tentando entender como criar a mão e anexar códigos (em C++ ou Python), sem nem se focar na rede neural que era a parte mais importante. Então teve-se que procurar uma nova solução para a simulação. Esta solução veio com o uso do Blender.

3.1 Simulação de Mão 3D no Blender

A solução escolhida foi criar manualmente uma mão 3D com armadura no Blender, ao invés de utilizar uma biblioteca. Seguiu-se os passos para criar uma armadura de acordo com o tutorial do youtube de Jonathan Williamson que mostra a criação do projeto CGC Classic: Rigged Fingers, que pode ser encontrado e baixado no site https://www.blendswap.com/blend/22357.

Tendo a mão 3D, utilizou-se as armaduras das juntas dos ossos (chamadas internamente de "Bone.019" para o polegar, "Bone.003" para o indicador, "Bone.007" para o médio, "Bone.019" para o anelar e o "Bone.015" para o mindinho) no modo de interação do objeto “Pose Mode” para gerar posições aleatórias dos dedos através da rotação no eixo X. As variáveis location.x/y/z, scale.x/y/z e rotation.w/y/z de cada osso estão “trancadas”, dessa forma a única variável que interfere na posição da mão é a rotação em relação ao eixo X, e é ela que será a entrada da Rede Neural criada. O código para gerar posições aleatórias no Blender é o *Código 1* do anexo.

Vale lembrar que *bpy.context.object.pose.bones["Bone.XXX"].rotation_quaternion[1]* é o código do Blender para a rotação do osso XXX, e o 1 do final representa o eixo X.

Decidiu-se utilizar A rotação desse osso, ao invés de uma armadura no ponto de cada dedo que indica as posições x, y e z, pois da segunda forma haveriam 15 variáveis de entrada, tornando a criação da rede neural ainda mais complexa. Esse seria um passo para o futuro.

3.2 Criação do Banco de Dados

Por conseguinte, o próximo passo foi a criação de uma base de dados para servir como entrada do treinamento da rede neural.

Foram escolhidas três preensões para a análise:

- *Preensão de Força Palmar – Grande Diâmetro* (Apêndice B – Imagem 1)

- Utilizou-se um cilindro de raio igual a 0.13 (em unidades do Blender) para chegar nessa posição.
- Preensão utilizada nas atividades de segurar um copo, agarrar uma maçã, agarrar uma lata, segurar no corrimão das escadas, segurar um secador de cabelo, segurar uma bola de handebol.

- *Preensão de Força Palmar – Pequeno Diâmetro* (Apêndice B – Imagem 2)

- Utilizou-se um cilindro de raio 0.070 (em unidades do Blender) para chegar nessa posição.
- Preensão utilizada nas atividades agarrar um guiador de bicicleta, agarrar o volante do carro, segurar uma vassoura, segurar uma escova de cabelo, agarrar um puxador de um móvel

- *Mão Fechada* (Apêndice B – Imagem 3)

- Rotacionou-se os dedos na direção da palma da mão, até o máximo para simular a mão fechada.
- Representará a classificação de objetos muito finos e pequenos, como um lápis, um batom, uma escova de dentes, etc.

Assim, gerou-se quarenta posições diferentes através do código mencionado anteriormente, e cada uma delas foi relacionada com uma das três preensões acima. A tabela do banco de dados segue no Apêndice C.

3.3 Criação da Rede Neural

No Spyder (Python 3.8) criou-se uma rede neural Perceptron Multicamadas na fase de treinamento. O número de camadas ocultas se deu pela média da soma das camadas de entrada e as de saída:

$$camadasOcultas = \frac{camadasEntrada + camadasSaida}{2}$$

Logo, no nosso caso, o número de camadas ocultas é 4.

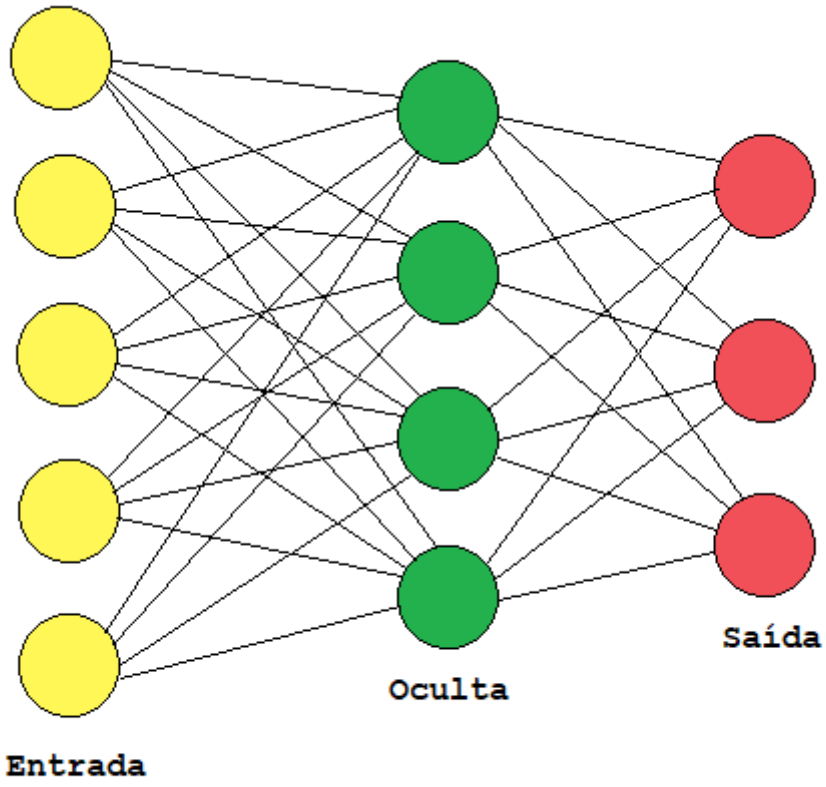


Figura 4. Representação das Camadas da Rede Neural Criada

Inicialmente, importamos o módulo numpy e o módulo pandas. O primeiro é utilizado realizar cálculos em Arrays Multidimensionais, e o segundo para manipulação e análise de dados permitindo importar o banco de dados criado que está em um arquivo .csv. Depois definimos a função de ativação e sua derivada, pois é usada no cálculo do delta. A função utilizada foi a função Sigmoid anteriormente apresentada. No código:

```
import numpy as np
import pandas as pd

def sigmoid(soma):
    return 1/(1+np.exp(-soma))

def sigmoidDeriv(sig):
    return sig * (1-sig)
```

Depois realiza-se a leitura do Banco de Dados gerado pelas posições da mão do Blender através da biblioteca pandas, que permite a leitura de arquivos .csv. A variável ‘entrada’ recebe os valores das rotações dos ossos, e a variável ‘saída’ recebe as saídas esperadas.

```
entradas = pd.read_csv("D:/Users/aless/Documents/UniCEUB/PIC/Rede Neural/TesteEntrada.csv", sep=";", encoding='ISO-8859-1')

saidas = pd.read_csv("D:/Users/aless/Documents/UniCEUB/PIC/Rede Neural/SaidasEsperadas.csv", sep=";", encoding='ISO-8859-1')
```

O próximo passo é inicializar as variáveis ‘epocas’, que informa quantas vezes será realizado o treinamento, ‘txAprend’, que é a taxa de aprendizado e será utilizada no cálculo dos novos pesos, ‘moment’, que é o momento e também é utilizado no cálculo dos novos pesos, e gerar pesos aleatórios dada pela função np.random.random().

```
epocas = 100000

txAprend = 0.3

moment = 1

pesos0 = 2*np.random.random((5,4)) - 1

pesos1 = 2*np.random.random((4,3)) - 1
```

Os argumentos da função np.random.random() são a quantidade de neurônios que tem nas duas camadas que ela conecta. Como pesos0 representa os pesos da camada de

entrada para a camada oculta, os parâmetros serão 5 (da camada de entrada) e 4 (da camada oculta). A mesma coisa ocorre em pesos1, porém entre a camada oculta e a camada de saída.

Agora inicia-se o treinamento da rede neural, que se repete até a variável j ficar igual ao número de épocas. Primeiramente, realiza-se o *Feedforward*, multiplicando as entradas pelos respectivos pesos em relação a cada neurônio da camada oculta, e aplicando a função Sigmoid nos resultados. Depois, multiplica-se as respostas dessa função pelos respectivos pesos que as interligam com a última camada, aplica-se a função Sigmoid novamente e chega-se na saída.

```
# --------Feedfoward-------

camadaEntrada = entradas

somaSinapse0 = np.dot(camadaEntrada,pesos0)

camadaOculta = sigmoid(somaSinapse0)


somaSinapse1 = np.dot(camadaOculta, pesos1)

camadaSaida = sigmoid(somaSinapse1)
```

Como é uma função multicamadas, é preciso aplicar o algoritmo de *backpropagation* (regra delta generalizada). Inicia-se encontrando o erro da saída. Nessa rede a função do erro quadrático médio foi definida como a função representativa do erro de aproximação:

$$f(x) = \frac{1}{n}\sum_{j=1}^{n}(d_j(k) - Y_j(k))^2 \qquad (7)$$

Em que Y_j (k) é o valor produzido pelo j-ésimo neurônio de saída da rede considerando-se a k-ésima amostra de treinamento, e d_j (k) é seu respectivo valor esperado. No código fica:

```
erro = saidas - camadaSaida

erroNeuronio = np.square(erro)

erroCamadaSaida = np.sum(erroNeuronio, axis=1)

mediaAbs = np.mean(erroNeuronio, axis = 1)

mediaErro = np.mean(mediaAbs)
```

Média erro representa o erro total da k-ésima época.

Agora os deltas são calculados para utilizá-los depois. Existirão dois deltas, um para a camada de saída e um para a camada oculta. O delta da camada de saída é simplesmente a multiplicação dos erros pela derivada da Sigmoide. Já o delta da camada oculta é a multiplicação dos pesos relativos à camada oculta e à camada de saída pela derivada da Sigmoide, multiplicado pelo delta da camada de saída.

```
derivadaSaida = sigmoidDeriv(camadaSaida)
deltaSaida = erro * derivadaSaida
```

Contudo, precisa-se da matriz transposta dos pesos para conseguir multiplicar, já que esta-se realizando multiplicações de matrizes.

```
pesos1Transp = pesos1.T
deltaSaidaXPeso = deltaSaida.dot(pesos1Transp)
deltaCamadaOculta = deltaSaidaXPeso*sigmoidDeriv(camadaOculta)
```

Por último serão calculados os novos pesos. O cálculo é dado pela multiplicação dos pesos antigos pelo momento, somado à multiplicação da camada que vem antes (no feedforward) pela taxa de aprendizagem e pelo delta da camada que vem depois.

Novamente precisa-se da matriz transposta para realizar a multiplicação.

```
camadaOcultaTransp = camadaOculta.T
pesosNovo1 = camadaOcultaTransp.dot(deltaSaida)
pesos1 = (pesos1 * moment) + (pesosNovo1 * txAprend)

camadaEntradaTransp = camadaEntrada.T
pesosNovo0 = camadaEntradaTransp.dot(deltaCamadaOculta)
 pesos0 = (pesos0 * moment) + (pesosNovo0 * txAprend)
```

4. Resultados e Discussão

Após realizar o treinamento da rede neural criada, chega-se aos seguintes resultados:

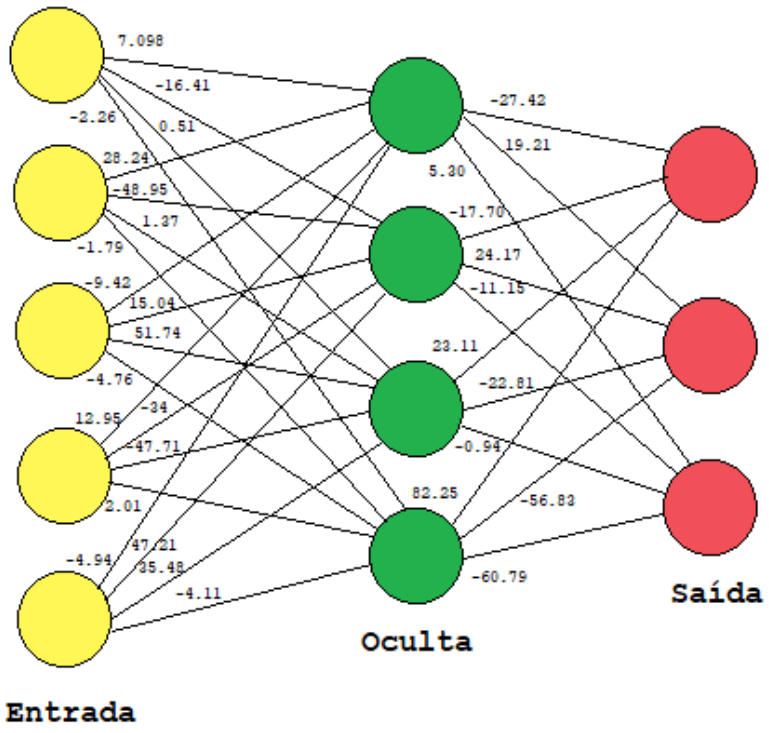
Pesos da camada de entrada para a camada oculta:

Index	0	1	2	3
Polegar	7.09873	-16.4101	0.513133	-2.25702
Indicador	28.2391	-48.952	1.36559	-1.79317
Médio	-9.42159	15.041	51.7385	-4.76065
Anelar	12.9512	-34.0036	-47.7173	2.00577
Mindinho	-4.93924	47.2177	35.4771	-4.10681

Pesos da camada oculta para a camada de saída:

	0	1	2
0	-27.4188	19.2086	5.30286
1	-17.697	24.1654	-11.1451
2	23.1136	-22.8131	-0.938589
3	82.2492	-56.8322	-60.7917

Para melhor entendimento:



Outra informação relevante que retira-se do código é o erro total. Ou seja, em quantos por cento a saída dada pelo código está diferenciando da saída ideal. Se o erro mostrado for maior que a porcentagem de erro aceitável, realiza-se o treinamento de novo, e tem-se resultados diferentes, pois os pesos iniciais são sempre aleatórios.

Neste caso o erro é a média dos erros quadrados médios, que deu 0.0206, ou aproximadamente 2%. Como a grande parte dos erros ficaram pequenos, o erro total também ficou pequeno.

5. Considerações finais

A pesquisa foi um grande desafio, pois em diversos momentos problemas surgiam que era preciso arranjar uma nova maneira de resolvê-los. O objetivo principal do projeto era conseguir criar uma rede neural que classificasse as posições das mãos em três preensões distintas, utilizando as coordenadas dos dedos. Apesar de ter-se chegado ao resultado final, não foi possível utilizar as coordenadas idealizadas, pois assim a rede criada teria que apresentar quinze neurônios na camada de entrada, o que tornaria a RNA muito mais complexa.

É possível ver que, mesmo com uma rede neural mais simples que a objetivada, foi possível realizar o reconhecimento de pressões, e assim, de objetos, o que ajuda no aumento da aceitação das próteses, mesmo que pequeno. Se fosse criado uma rede neural mais complexa, com mais sensores e variáveis, seria possível chegar a um ponto no qual a prótese reage igual à uma mão?

É possível continuar essa pesquisa, tentando encontrar uma forma de melhorar a rede neural criada e utilizar, no local da rotação, as coordenadas dos dedos das mãos, representando os sensores de toque e pressão. Outra área onde pode-se utilizar essa rede neural ou uma semelhante é na modelagem de próteses de membros inferiores, para o reconhecimento do material onde se pisa (por exemplo, areia, madeira, lama) e assim aplicar a força necessária para mover-se naquele terreno.

Referências

Nordin, M.; Frankel, V. H. *Biomecânica do sistema musculoesquelético*. 3 ed. Rio de janeiro: Guanabara Koogan, 2003.

Carvalho, G. L. de. *Proposta de um método de projeto de próteses de membros superiores com a utilização da engenharia e análise do valor*. 2004. Tese de Doutorado. Universidade de São Paulo.

Kapandji, A. I. *Fisiologia Articular: esquemas comentados de mecânica humana*. v.1, 6 ed. Rio de Janeiro: Guanabara Koogan, 2007.

Costa, R. M. *Adaptação do usuário de próteses mioelétricas: implicações na aprendizagem dos movimentos da mão* - Tese de Doutorado apresentada ao Programa de Pós-Graduação em Biotecnologia da Rede Nordeste de Biotecnologia (RENORBIO) do Ponto Focal Espírito Santo da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do título de Doutora em Biotecnologia. Orientador: Prof. Dr. Teodiano Freire Bastos Filho Vitória. 2017. Tese (Doutorado) - Rede Nordeste de Biotecnologia (RENORBIO), Vitória, 2017.

Feix, T.; Romero, J.; Schmiedmayer, H.-B.; Dollar, A. M.; Kragic, D. *The GRASP Taxonomy of Human Grasp Types*. (2016). IEEE Transactions on Human-Machine Systems 46(1): 66-77

Pontifícia Universidade Católica do Rio de Janeiro - PUCRIO. *Redes Neurais Artificiais*. [S. l.], 201-?. Certificado digital nº 0711116/CA. Disponível em: https://www.maxwell.vrac.puc-rio.br/16580/16580_4.PDF. Acesso em: 20 out. 2020

Silva, I. N. da; Spatti, D. H.; Flauzino, R. A. *Redes Neurais Artificiais Para Engenharia e Ciências Aplicadas. Fundamentos Teóricos e Aspectos Práticos*. 2 ed. São Paulo: Artliber Editora, 2016.

Godoi, T. da S. M. de. *Prótese Mioelétrica Controlada por Redes Neurais.* Trabalho de Conclusão de Curso apresentado ao Centro Universitário de Brasília (UniCEUB), Brasília, 2013.

Szabó, B. K. *Rigged Hand Model for the Blender Game Engine.* Hungarian Academy of Sciences Centre for Energy Research, Budapest, Hungary. Faculty of Informatics, University of Debrecen, Debrecen, Hungary. Disponível em: https://ojs.lib.unideb.hu/rIim/article/view/3916/3794. Acesso em: 18 out. 2020

Anexos:

Anexo A: Código da mão 3D no Blender

Criado por Jonathan Williamson (cgcookie), e pode ser encontrado para download no site: https://www.blendswap.com/blend/22357

Apêndices:

Apêndice A: Código Gerar Posições Aleatórias Blender

```
import bpy
import os
from random import uniform

polegar = 0
indicador = 0
medio = 0
anelar = 0
mindinho = 0

polegar = uniform(-0.170, 0.431)
indicador = uniform(0, 0.429)
medio = uniform(0, 0.845)
anelar = uniform(0, 0.821)
mindinho = uniform(0, 0.520)

bpy.context.scene.frame_set(20)
#Polegar
bpy.context.object.pose.bones["Bone.019"].rotation_quaternion[1] = -polegar
#Indicador
bpy.context.object.pose.bones["Bone.003"].rotation_quaternion[1] = -indicador
#Médio
bpy.context.object.pose.bones["Bone.007"].rotation_quaternion[1] = -medio
#Anelar
bpy.context.object.pose.bones["Bone.011"].rotation_quaternion[1] = -anelar
#Mindinho
bpy.context.object.pose.bones["Bone.015"].rotation_quaternion[1] = -mindinho
```

Apêndice B: Imagens das Posições das Mãos no Blender

Imagem 1 – Grande Diâmetro

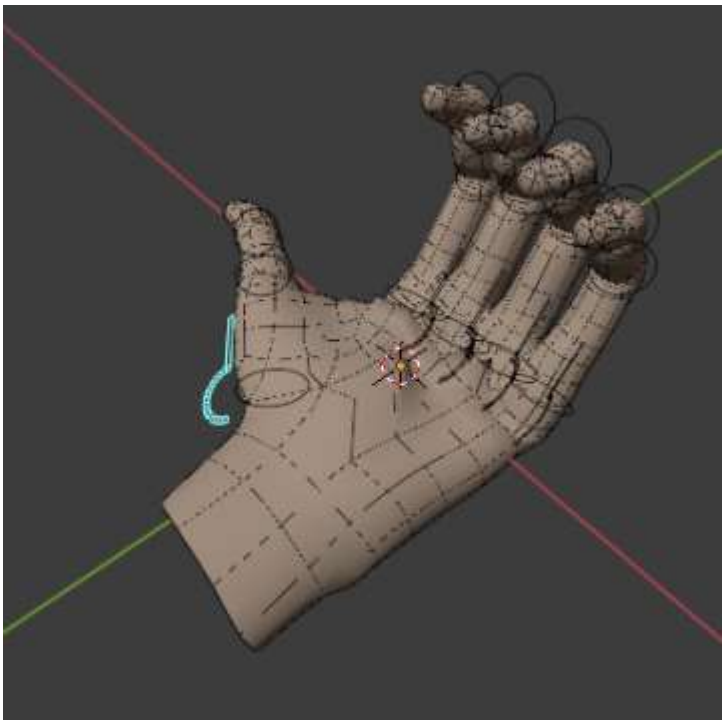


Imagem 2 – Pequeno Diâmetro

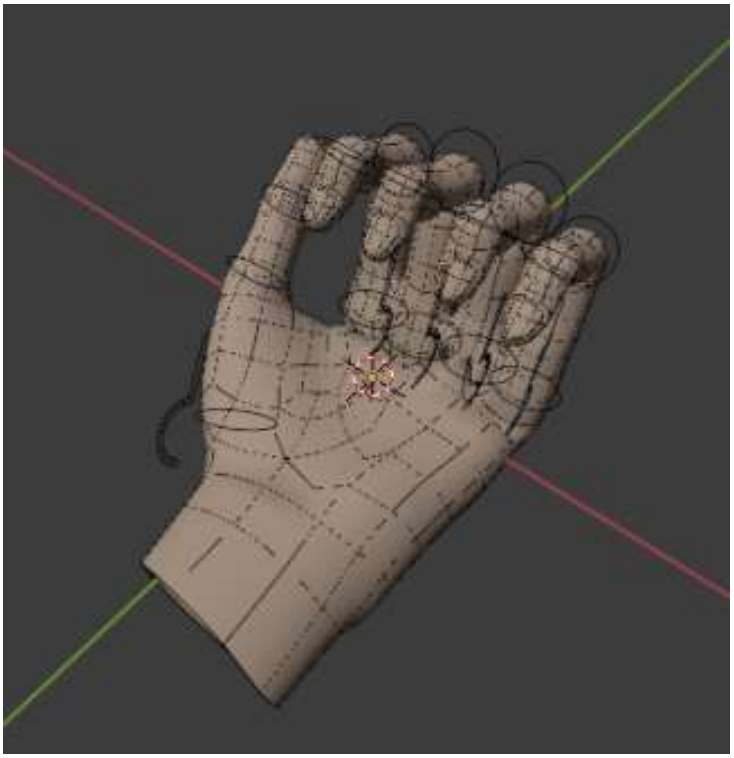
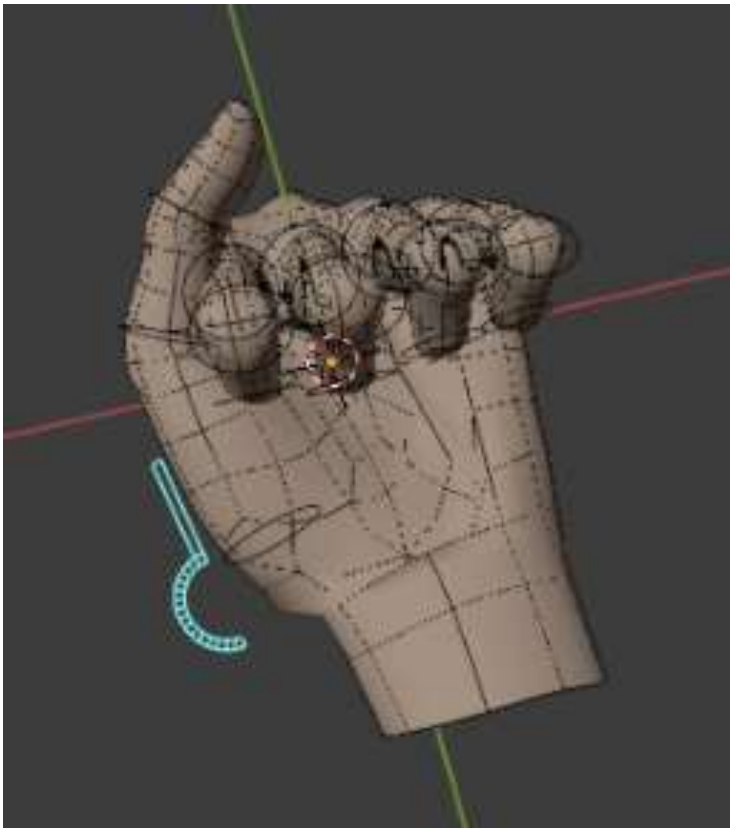


Imagem 3 – Mão Fechada



Apêndice C: Banco de Dados

Entradas:

Polegar	Indicador	Médio	Anelar	Mindinho
0.25278	0.3052	0.45213	0.252618	0.32724
-0.11243	0.1152	0.062295	0.2793	0.093452
-0.00537	0.029417	0.0918	0.710272	0.352324
0.250542	0.359014	0.575958	0.631309	0.266173
-0.13938	0.294095	0.170731	0.371173	0.182229
0.207029	0.06084	0.102639	0.772277	0.422332
0.124063	0.28419	0.364276	0.679864	0.416591
0.165401	0.287602	0.394582	0.095156	0.30502
0.20475	0.04837	0.41602	0.686476	0.342983
0.14137	0.312092	0.13534	0.217265	0.15453
-0.03557	0.095834	0.470288	0.777814	0.097606
0.10575	0.364411	0.175134	0.248403	0.294237
0.349021	0.215212	0.005208	0.346092	0.161814
-0.05562	0.044037	0.52301	0.790787	0.428155
-0.12553	0.241502	0.686861	0.565127	0.33833
0.269458	0.25844	0.289165	0.197032	0.494939
0.385933	0.23007	0.644365	0.457838	0.175906
0.050404	0.249964	0.454165	0.390038	0.353072
-0.12522	0.37137	0.527494	0.820382	0.355852
-0.12104	0.187394	0.081607	0.480588	0.037911
0.252987	0.296449	0.78482	0.63526	0.370831
-0.09859	0.16874	0.475976	0.402652	0.274696
-0.03672	0.05456	0.033022	0.740186	0.330718
0.304598	0.3824	0.508476	0.077284	0.167374
-0.0543	0.280795	0.16322	0.703381	0.301715
0.179413	0.3096	0.7555	0.535032	0.069579
-0.0151	0.22202	0.643017	0.793562	0.448224
-0.16343	0.14064	0.445195	0.80413	0.347805
0.19145	0.058335	0.529633	0.213367	0.093765
-0.04622	0.046235	0.154799	0.18462	0.024532
0.392005	0.219301	0.072805	0.145565	0.049621
0.256715	0.34613	0.77753	0.208095	0.258799
0.244148	0.423697	0.019318	0.310984	0.396959
0.161284	0.27496	0.826397	0.260877	0.101661
0.390472	0.308587	0.551726	0.72829	0.236086
0.321056	0.224184	0.173231	0.456634	0.24093
-0.11774	0.03739	0.184552	0.132632	0.146334
0.024038	0.225412	0.149588	0.178918	0.142504
0.126713	0.176148	0.375118	0.630688	0.08998
-0.02258	0.249003	0.215994	0.311905	0.424407

Saidas Esperadas:

d1	d2	d3
0	0	1
1	0	0
1	0	0
0	0	1
0	1	0
0	1	0
0	1	0
0	1	0
0	0	1
1	0	0
0	1	0
1	0	0
0	1	0
0	1	0
0	0	1
0	1	0
0	0	1
0	0	1
0	0	1
1	0	0
0	0	1
0	1	0
1	0	0
0	0	1
0	1	0
0	0	1
0	0	1
0	1	0
1	0	0
1	0	0
0	1	0
0	0	1
0	0	1
0	0	1
0	0	1
0	1	0
1	0	0
1	0	0
0	1	0
0	1	0

Com d1, d2, d3 representando os neurônios da camada de saída, e a codificação:

1	0	0	Preensão de Força Palmar – Grande Diâmetro
0	1	0	Preensão de Força Palmar – Pequeno Diâmetro
0	0	1	Mão Fechada