UNIVERSIDADE FEDERAL DE UBERLÂNDIA
Vítor Hugo Honorato Tiago

Predição de proteínas exportadas por meio de Redes Neurais Artificiais

Uberlândia
2020

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
Vítor Hugo Honorato Tiago

Predição de proteínas exportadas por meio de Redes Neurais Artificiais

Monografia apresentada ao Curso de Bacharelado em Ciência da Computação da Universidade Federal de Uberlândia, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

Universidade Federal de Uberlândia – UFU

Faculdade de Computação

Curso de Bacharelado em Ciência da Computação

Orientador: Anderson Rodrigues dos Santos

Uberlândia

2020

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
Vítor Hugo Honorato Tiago

Predição de proteínas exportadas por meio de Redes Neurais Artificiais

Monografia apresentada ao Curso de Bacharelado em Ciência da Computação da Universidade Federal de Uberlândia, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

Trabalho aprovado. Uberlândia, 2020:

Anderson Rodrigues dos Santos
UFU

Maria Adriana Vidigal de Lima
UFU

Márcia Aparecida Fernandes
UFU

Uberlândia
2020

"A utopia está lá no horizonte. Me aproximo dois passos, ela se afasta dois passos. Caminho dez passos e o horizonte corre dez passos. Por mais que eu caminhe, jamais alcançarei. Para que serve a utopia? Serve para isso: Para que eu não deixe de caminhar"

(Fernando Birri)

Resumo

Uma das principais formas de combate a doenças é a vacinação. Dentre as etapas da produção de uma vacina é necessária a escolha de proteínas que farão parte da mesma. O objetivo central do trabalho é identificar proteínas exportadas. Essas possuem maior chance de serem efetivas quando utilizadas em vacinas contra bactérias patogênicas. Assim, utilizamos técnicas de Inteligência Artificial, em especial Redes Neurais Artificiais para a classificação de proteínas em exportadas e não exportadas. Para alcançar tal objetivo, primeiramente construimos uma base de dados com proteínas extraídas dos genomas de 35 espécies de bactérias patogênicas, utilizada posteriormente para treinamento da rede neural. Em uma segunda etapa, implementamos uma rede neural utilizando-se do algoritmo *backpropagation* para treinamento, capaz de classificar essa base de dados. Com isso, realizamos várias execuções com diferentes arquiteturas de rede e diversos parâmetros de treinamento. Comparamos os resultados obtidos das execuções da rede implementada com os resultados do Weka. Utilizamos dez indíces de propensão (PARJ860101, JOND750101, EISD840101, JURD98010, Base, Ácido, Polar, Não Polar, Massa e Massa Molar). Com isso, a rede neural obteve uma taxa de acerto de 82.1%, enquanto o Weka atingiu uma taxa de acerto de 87.2%. Os resultados são promissores, visto que a rede neural implementada teve taxa de acerto próxima ao Weka e ainda, mesmo sem otimizações, conseguiu finalizar seu treinamento em um tempo menor que o algoritmo do Weka em algumas execuções, demorando 4.7% menos tempo para realizar o treinamento. Implementamos a rede neural utilizando a linguagem de programação C e essa está disponível para visualização em https://github.com/tiagohugovitor/Rede-Neural-TCC.

Palavras-chave: RNA's, backpropagation, proteínas, exportadas, vacinas, Weka.

Abstract

One of the main ways of fighting diseases is vaccination. One of the phases for producing a vaccine consists of choosing some proteins that will compose them. The central objective of this paper is to identify exported proteins. These proteins have more chances to be effective when utilized on vaccines against pathogenic bacteria. We utilized Artificial Intelligence techniques like Artificial Neural Networks to classify the proteins in exported or not exported. To achieve that goal, we constructed a database of proteins extracted from the genomes of 35 species of pathogenic bacteria, utilized to train the network. After that, we implemented an Artificial Neural Network with training by the backpropagation algorithm to classify the proteins database. We prepared multiple executions with differents architectures of the network and diverse training parameters. We compared the results of these executions from the implemented network with the results of Weka. We employed ten propensity indexes (PARJ860101, JOND750101, EISD840101, JURD98010, Basic, Acid, Polar, NonPolar, Mass and Molar mass). Then, the network had a hit rate of 82.1%; meanwhile, Weka had a hit rate of 87.2%. The results were promissory, analyzing that the artificial neural network implemented had an accuracy close to the Weka. Even without optimization, the network finalized it is training on time shorter than Weka's algorithm on some executions, taking 4.7% less time to train. We implemented the artificial neural network using the language C, and the code is available on https://github.com/tiagohugovitor/Rede-Neural-TCC.

Keywords: ANN's, backpropagation, proteins, exported, vaccines, Weka.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

RNA	Rede Neural Artificial
IC	Inteligência Computacional
IA	Inteligência Artificial

Sumário

1 Introdução

Define-se como patogenicidade, a capacidade de um agente biológico, conhecido como patógeno, causar doenças em seu hospedeiro (ROBERTIS; HIB, 2006). Uma das principais formas de combate e prevenção de doenças é a vacinação. Dentre as várias etapas na produção de vacinas contra agentes patogênicos, as fases iniciais consistem no isolamento de proteínas produzidas pelo próprio patógeno. Para um funcionamento efetivo da vacina, as proteínas isoladas devem conter partes responsáveis por ativar o sistema imunológico do hospedeiro e produzir anticorpos (REZENDE et al., 2016).

O alvo de estudo desse trabalho são células bacterianas patogênicas. Células bacterianas produzem proteínas que podem ser classificadas em quatro tipos, relacionados ao seu direcionamento após a produção: Citoplasmáticas, Membranares, Potencialmente Expostas e Exportadas (KHANACADEMY, 2019). Proteínas são uma sequência de aminoácidos que possuem uma função específica dentro da célula.

As células bacterianas patogênicas invadem seu hospedeiro e secretam algumas proteínas em tal organismo. Essas proteínas são denominadas exportadas e são responsáveis principalmente pela adesão, colonização e liberação de exotoxinas no hospedeiro. Devido a essas funções, as proteínas exportadas possuem uma alta probabilidade de ativar o sistema imunológico humano. Por isso, são as principais utilizadas na produção de vacinas contra o agente patogênico (SANTOS et al., 2013).

Entretanto, há uma grande complexidade na identificação de quais proteínas serão exportadas, visto que essas são uma pequena proporção de todas proteínas produzidas pela célula. Sendo assim, é válida a construção de um sistema computacional que, dada a sequência de aminoácidos de uma determinada proteína, consiga identificar se ela será exportada.

A Bioinformática é uma ciência multidisciplinar que utiliza de técnicas computacionais e análises estatísticas para resolução de problemas relacionados a biologia (ALVES, 2013). O trabalho apresentado a seguir se enquadra no contexto de bioinformática, ao utilizar Redes Neurais Artificiais para reconhecimento e classificação de proteínas exportadas em células bacterianas.

O estudo de *Redes Neurais Artificiais*(RNA's) é uma das principais vertentes de *Inteligência Computacional(IC)*, um ramo da área de *Inteligência Artificial*(IA). A Inteligência Computacional se preocupa em implementar ações inteligentes em agentes computacionais. Por sua vez, RNA's são modelos computacionais inspirados no cérebro humano e utilizados com diversas finalidades, entre elas, destaca-se o uso para reconhecimento de padrões (RUSSELL; NORVIG, 2013).

Com isso, o objetivo do trabalho é implementar uma rede neural que consiga reconhecer padrões nas sequências de aminoácidos das proteínas exportadas, limitando radicalmente o conjunto de proteínas para busca por partes que disparem o sistema imunológico humano (BENDTSEN et al., 2005).

É importante destacar que o preditor construido através da rede neural será utilizado para substituir a ferramenta SurfG, que prediz localização subcelular de proteínas, em um trabalho coordenado pelo professor orientador.

Esse trabalho está estruturado da seguinte forma. No capítulo 2, é apresentado uma fundamentação teórica necessária para melhor entendimento do assunto. No capítulo 3, é explicada a metodologia e feito o detalhamento das implementações realizadas. No capítulo 4 são exibidos os experimentos e analisados os resultados encontrados. E, finalmente, no capítulo 5, são feitas conclusões e sugestões de trabalhos futuros.

2 Referencial Teórico

Nesse capítulo é construído um referencial teórico com o intuito de explicar e contextualizar o leitor acerca de todos os conceitos científicos utilizados no presente trabalho.

2.1 Proteínas

Nas células existem três polímeros responsáveis pelas principais funções celular: os ácidos nucléicos, os polissacarídeos e as proteínas (ROBERTIS; HIB, 2006). O foco do trabalho são proteínas produzidas em células bacterianas patogênicas. Em sua estrutura, proteínas são cadeias de aminoácidos de qualquer tamanho, construídas para realizar uma função específica dentro ou fora da célula.

2.1.1 Aminoácidos

Os aminoácidos são os componentes básicos de uma proteína. Existem somente vinte aminoácidos utilizados na produção de proteínas, entretanto as possibilidades de combinação, ordenação e quantidade de tais aminoácidos permitem uma vasta quantidade de proteínas existentes. Os vinte aminoácidos podem ser classificados em quatro grupos: Ácidos, Básicos, Neutros Polares e Neutros Apolares (ROBERTIS; HIB, 2006). Na figura 1 estão especificados todos os aminoácidos com sua classificação nos grupos acima citados.

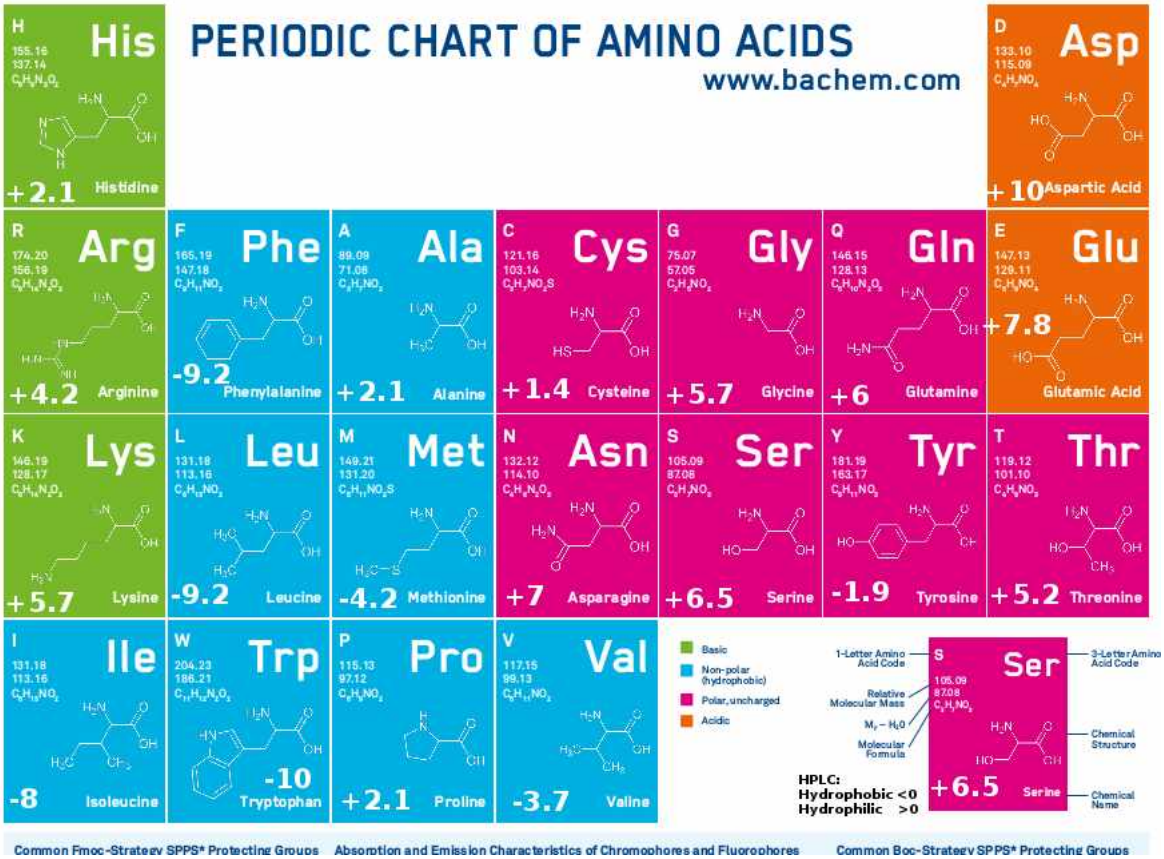


Figura 1 – Tabela periódica dos aminoácidos

Fonte: Figura adaptada pelo autor encontrada em www.bachem.com

A classificação dos aminoácidos é feita através de seu pH, sendo que existem ácidos, básicos e os neutros. Os neutros são divididos entre neutros polares, possuem afinidade com a água e neutros apolares, não possuem afinidade com a água. Na figura 1 os aminoácidos básicos são representados em verde, os neutros apolares em azul, os neutros polares em rosa e os ácidos em laranja.

Além da classificação dos aminoácidos, a figura 1 também apresenta várias características específicas de cada aminoácido. Essas e outras características são utilizadas posteriormente para identificação e representação de proteínas no treinamento da rede neural construída.

2.1.2 Peptídeos Sinais

As proteínas nem sempre são produzidas no mesmo local em que elas devem exercer sua função. Uma vez que elas podem exercer funções em qualquer parte do interior ou exterior da célula é necessário que a informação da localidade final dessa proteína esteja definida em alguma parte da mesma. O peptídeo sinal é uma subcadeia de 3 a 70 aminoácidos presente em uma proteína para indicar o local para o qual a proteína deve ser transportada e exercer sua função (KAPP et al., 2009).

2.1.3 Classificação de Proteínas

As proteínas produzidas dentro de uma célula podem ser classificadas em quatro tipos, quanto ao seu local final de atuação. Esses tipos são:

- Citoplasmáticas
- Membranares
- Potencialmente Expostas
- Exportadas

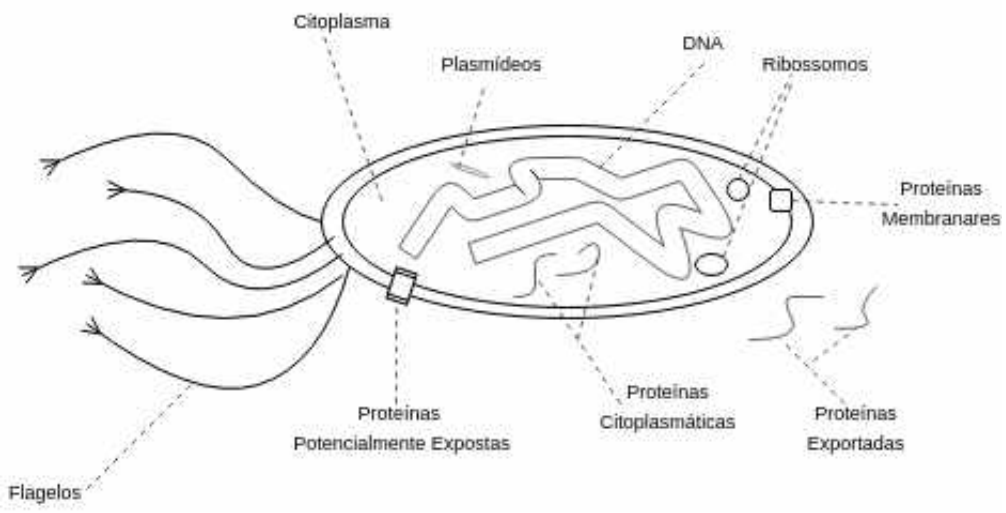


Figura 2 – Modelo de bactéria procarionte

Fonte: Figura construída pelo autor

A figura 2 representa uma bactéria procarionte e indica os locais de ativação das proteínas produzidas pela mesma.

As proteínas citoplasmáticas tem sua produção e atuação estritamente no interior da célula. Já as proteínas membranares estão localizadas na membrana celular podendo estar em contato apenas com o interior da célula. Por sua vez, as proteínas do tipo potencialmente expostas também estão localizadas na membrana celular, entretanto possuem contato com o exterior da célula (KHANACADEMY, 2019).

O alvo principal desse trabalho são as proteínas do tipo exportadas. As proteínas exportadas depois de produzidas são transportadas e liberadas completamente no exterior da célula. No caso específico de células bacterianas patogênicas essas proteínas são as responsáveis pela adesão, colonização e liberação de exotoxinas no seu hospedeiro. Sendo assim, as proteínas exportadas pela célula induzem o sistema imunológico do hospedeiro na produção de anticorpos contra o patógeno que o está infectando. Portanto, tais proteínas são as mais frequentemente utilizadas na produção de vacinas.

2.2 Inteligência Computacional(IC)

Inteligência Computacional é uma subárea de Inteligência Artifical. A IC tem como foco lidar com problemas que possuem um alto teor de imprecisão. Para isso, são utilizados algoritmos flexíveis que são capazes de se adaptar a contextos ou realizar técnicas de aprendizado, sempre se aperfeiçoando. Os principais campos de estudo em IC são: Computação Evolutiva, Redes Neurais Artificiais e Lógica Fuzzy (RUSSELL; NORVIG, 2013).

No presente trabalho foram utilizadas redes neurais para classificação das proteínas, usufruindo de sua aplicação em reconhecimento de padrões.

2.2.1 Redes Neurais Artificais (RNA's)

Redes Neuras Artificiais são modelos computacionais que procuram entender e utilizar a atividade do cérebro humano, em específico sua capacidade de aprender. As RNA's se baseiam no fato das atividades cerebrais humanas consistirem em uma sequência de interações eletroquímicas entre células denominadas neurônios (RUSSELL; NORVIG, 2013). Um dos principais diferenciais das redes neurais é sua capacidade de aprendizagem, conseguindo se adaptar para resolver diversos tipos de problemas. Uma dentre suas várias aplicações é o reconhecimento de padrões (ZHANG, 2000), onde uma rede é construída e treinada para classificar sua entrada em algum dos subconjuntos de saída.

2.2.1.1 Neurônio Artificial

O neurônio é a unidade básica responsável pelo funcionamento do sistema nervoso humano. A sua principal função é receber, processar e propagar impulsos elétricos.

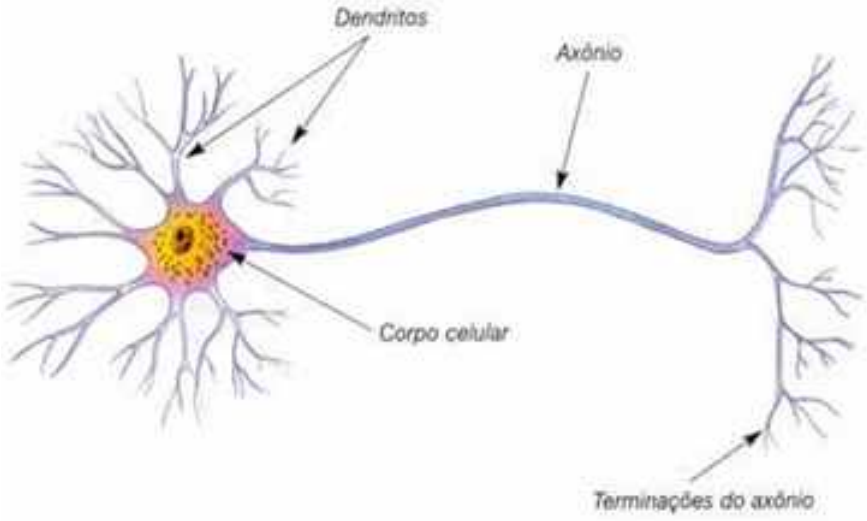


Figura 3 – Modelo de um neurônio biológico

Fonte: Figura encontrada em https://imasters.com.br/desenvolvimento/redes-neurais-roots-parte-01

Na figura 3 é representado um modelo de neurônio humano. O neurônio é composto principalmente por três partes distintas: os dendritos, o axônio e o corpo celular. Os dendritos são ramificações finas responsáveis por captar impulsos elétricos do ambiente ou de outras células e conduzi-los ao corpo celular. O corpo celular é a região onde se localiza o núcleo da célula nervosa, ela é responsável principalmente por processar os impulsos capturados pelos dendritos e decidir quais estímulos devem ser repassados. Por sua vez, o axônio recebe impulsos do corpo celular e repassa esses impulsos distribuindo-os para o ambiente e para outras células (KHANACADEMY, 2020).

Baseando-se no neurônio humano biológico, foi desenvolvido em 1943 por McCulloch e Pitts o primeiro modelo matemático para simulação artificial de um neurônio (RUSSELL; NORVIG, 2013). Esse modelo é ilustrado na figura 4.

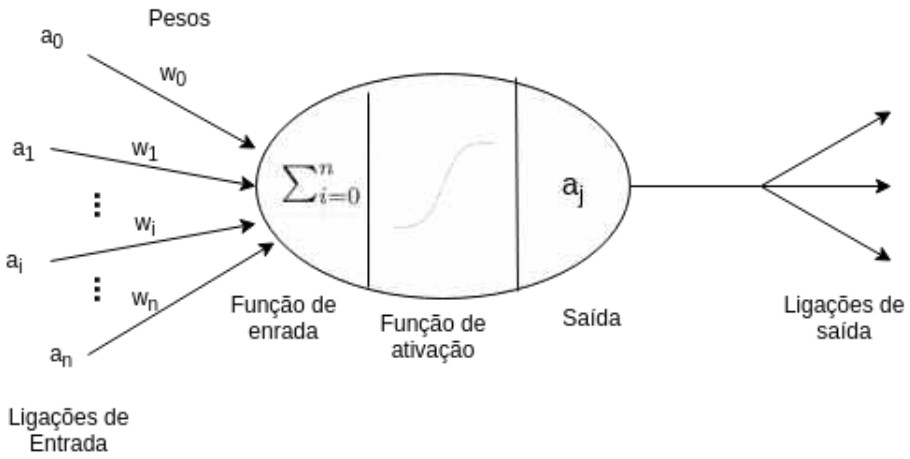


Figura 4 – Modelo matemático de um neurônio

Fonte: Figura refeita pelo autor baseado em (RUSSELL; NORVIG, 2013) página 635

O neurônio artificial é a unidade básica de uma Rede Neural Artificial. Um neurônio artificial é composto por três partes: a entrada dos dados, a função de ativação e a saída dos dados. É possível traçar um paralelo entre o neurônio artificial e seu correspondente biológico, assim a entrada dos dados caracteriza os dendritos, a função de ativação representa o corpo celular e por fim a saída dos dados é equivalente ao axônio.

Na camada mais externa de um neurônio artificial ocorre a entrada dos dados que serão tratados. Na figura 4 os dados de entrada são representados pelas variáveis $a_0, a_1, ..., a_i, ..., a_n$ e cada aresta de entrada do neurônio possui um peso, representados pelas variáveis $w_0, w_1, ..., w_i, ..., w_n$. Os pesos são atribuídos a ligação de uma aresta com um neurônio e indicam a importância do dado proveniente daquela aresta no processamento realizado pelo neurônio.

Em especial, a entrada a_0 é geralmente 1 e seu peso é denominado o bias do neurônio. O Bias é responsável por adicionar uma constante ao modelo do neurônio. Dessa

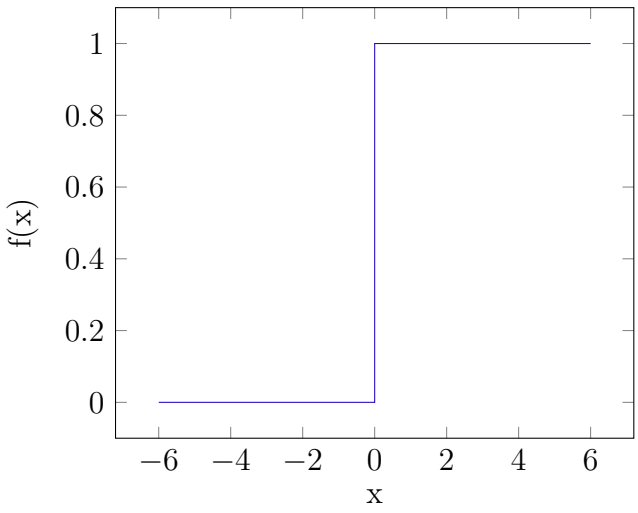
maneira, ele ajuda a rede a se adaptar melhor para fornecer os resultados esperados, visto que, sendo a_0 sempre 1, ele é independente da entrada dos dados.

A camada de entrada possui uma função responsável por processar os dados de entrada com seus determinados pesos e transformá-los em um único valor. Na figura 4, a função de entrada é representada por um simples somatório do produto de a_i e w_i adicionado ao bias do neurônio, demonstrada na equação 2.1.

$$Entrada = \sum_{i=1}^{n}(a_i \cdot w_i) + bias \tag{2.1}$$

A segunda camada presente no modelo de um neurônio artificial é composta por uma função denominada função de ativação. A camada de ativação capta o valor recebido pela camada de entrada, aplica uma função de ativação nele e o envia para a camada de saída. É importante destacar a existência de dois tipos de funções de ativação: funções de limiar rígido e funções logísticas. As funções de limiar rígido possuem somente valores discretos na saída, enquanto funções logistícas atribuem valores contínuos na saída da função, sendo diferenciáveis (RUSSELL; NORVIG, 2013). Abaixo as figuras 5 e 6 representam os dois tipos de funções.

Figura 5 – Função de limiar rígido

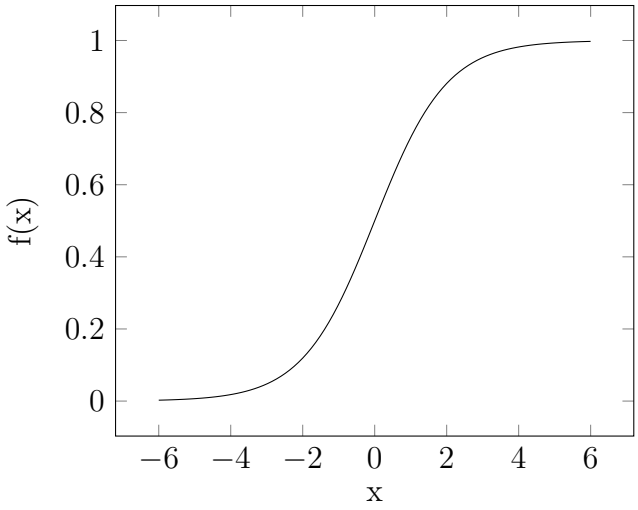


Fonte: Gráfico montado pelo autor

Na figura 5 é representada uma função de limiar rígido, cuja saída é 0 ou 1. Sua função é definida por:

$$f(x) = 1, \ se\ x > 0\ ou\ 0, \ se\ x < 0 \tag{2.2}$$

Figura 6 – Função logística sigmóide



Fonte: Gráfico montado pelo autor

Na figura 6 é representada uma função logística, mais conhecida como sigmóide, sua equação é:

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.3}$$

Note que a função de limiar rígido não é diferenciável no ponto $x = 0$, enquanto a função sigmóide é contínua, e portanto diferenciável, em todos os seus pontos.

A terceira camada do neurônio artificial é conhecida como camada de saída. Essa camada tem como única responsabilidade direcionar o valor gerado na saída da função de ativação para a saída do neurônio.

2.2.1.2 Estrutura das Redes Neurais Artificiais

Uma Rede Neural Artificial é composta por vários neurônios interconectados por ligações direcionadas. Na figura 7 é representado o modelo de uma RNA.

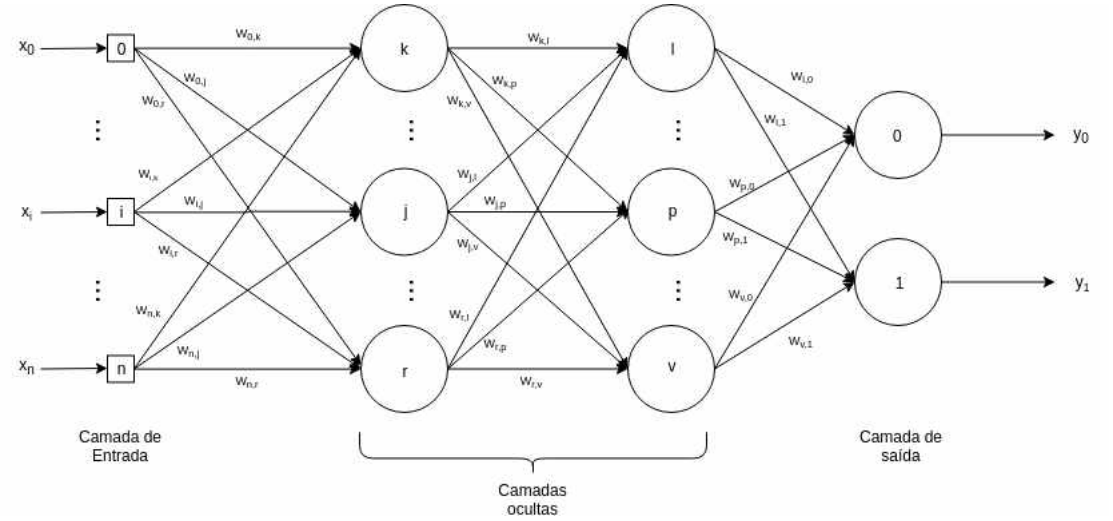


Figura 7 – Modelo de uma Rede Neural

Fonte: Figura construída pelo autor.

As Redes Neurais tem sua subdivisão feita em camadas. Existem três camadas representadas na figura 7: a camada de entrada, a camada oculta e a camada de saída. A camada de entrada são as unidades de dados que estão dispostas para serem inseridas no início da rede. Em paralelo, a camada de saída é caracterizada pelo(s) neurônio(s) que transportam os dados processados para a saída da rede neural, sendo esses os últimos neurônios a processar os dados. Uma RNA obrigatoriamente possui uma camada de entrada e uma camada de saída. Entre a camada de entrada e a camada de saída podem existir n camadas denominadas ocultas, sendo $n \geq 0$. As camadas ocultas são responsáveis estritamente pelo processamento dos dados. A Figura 8 *(a)* ilustra um exemplo de RNA sem camada oculta, em *(b)* um exemplo com uma única camada oculta, e por fim, em *(c)*, um exemplo com várias camadas ocultas.

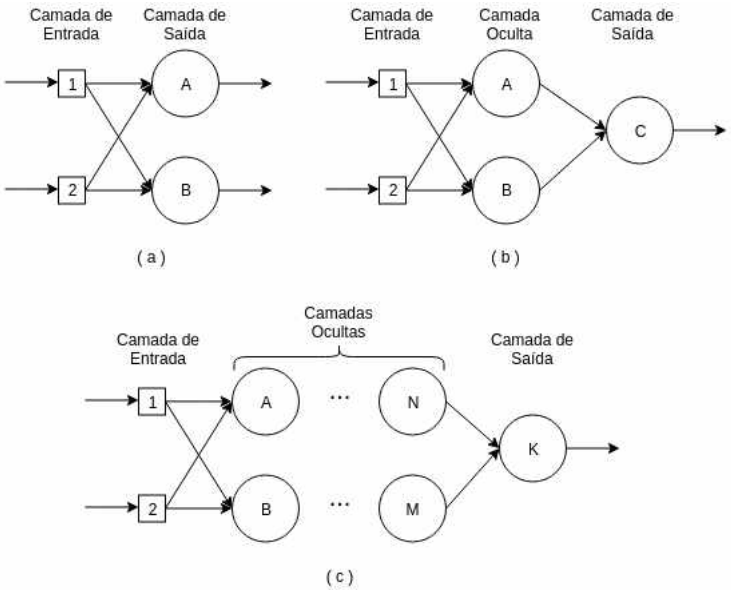


Figura 8 – Exemplos de Redes Neurais

Fonte: Figura construída pelo autor.

A figura 7 representa uma RNA com duas camadas ocultas. Na figura, os dados de entrada da rede são indicados pelas variáveis $x_0, ..., x_i, ..., x_n$. Os pesos das arestas têm sua nomenclatura relacionada ao neurônio em que o dado acabou de sair e para qual neurônio o dado será direcionado, portanto a aresta que liga o dado de saída do neurônio j até a entrada do neurônio p tem seu peso denominado $w_{j,p}$. A camada de saída da rede da figura 7 possui dois neurônios *0* e *1* e produzem as saídas indicadas por y_0 e y_1.

2.2.1.3 Execução da Rede Neural

A execução de uma rede neural é unidirecional, ou seja, os dados são colocados na entrada e perpassam por todos os neurônios, camada após camada, até serem conduzidos para a camada de saída. Na camada de saída encontra-se o resultado obtido através do processamento dos dados de entrada. A figura 9 representa uma rede neural com três neurônios na camada de entrada, duas camadas ocultas e dois neurônios na camada de saída. Tal rede da figura 9 é utilizada para exemplificar a execução de uma rede neural.

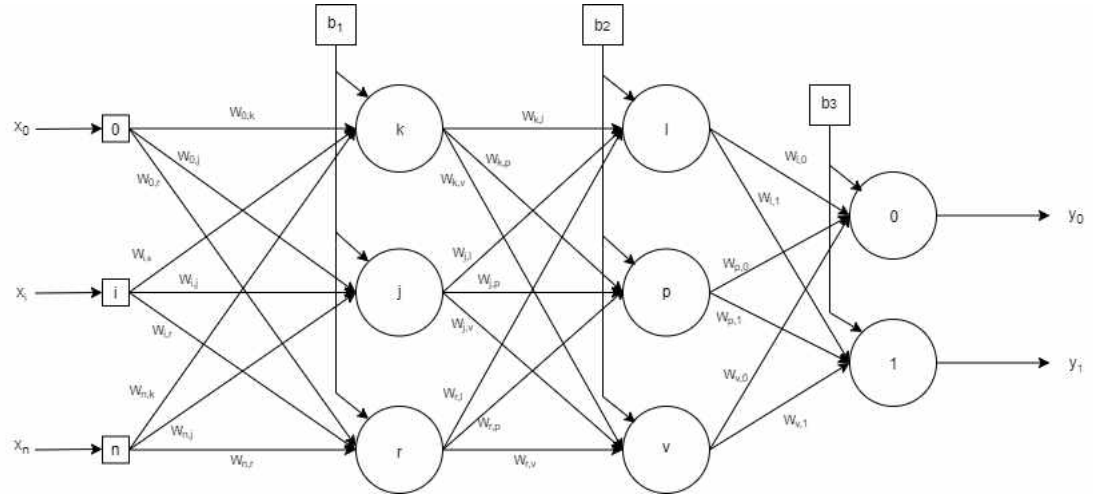


Figura 9 – Exemplo de Rede Neural

Fonte: Figura construída pelo autor.

O primeiro passo na execução da rede neural é a propagação dos dados de entrada na camada de entrada. Portanto, os dados x_0, x_i e x_n são colocados respectivamente nos neurônios de entrada *0*, i e n. Após isso, para cada neurônio da primeira camada oculta (k, j e r) é feito o processamento dos dados.

O neurônio k recebe em sua entrada os dados x_0, x_i e x_n e os pesos de suas arestas, respectivamente $w_{0,k}$ $w_{i,k}$ e $w_{n,k}$. Com isso, o neurônio k realiza a soma de suas entradas ponderadas com os seus pesos e acrescenta o valor do seu bias. Logo, a entrada de processamento do neurônio k é representada por:

$$E_k = (x_0 \cdot w_{0,k}) + (x_i \cdot w_{i,k}) + (x_n \cdot w_{n,k}) + b_1 \tag{2.4}$$

Em seguida, o valor de E_k é enviado a função de ativação do neurônio. Para o exemplo será utilizada a função de ativação sigmóide, portanto, a saída do neurônio k é dada por:

$$S_k = f(E_k) = \frac{1}{1 + e^{-E_k}} \tag{2.5}$$

Após o processamento em k, é feito o mesmo processamento em j e em r. Então, com todas as saídas dos neurônios da primeira camada oculta calculadas, o processamento passa para a próxima camada. Assim, os neurônios l, p e v começam seu processamento utilizando S_k, S_j e S_r como suas entradas.

O neurônio l recebe em sua entrada os dados já processados S_k, S_j e S_r e os pesos de suas arestas, respectivamente $w_{k,l}$, $w_{j,l}$ e $w_{r,l}$. Com isso, o neurônio l realiza a soma

de suas entradas ponderadas com os seus pesos e acrescenta o valor do seu bias. Logo, a entrada de processamento do neurônio l é representada por:

$$E_l = (S_k \cdot w_{k,l}) + (S_j \cdot w_{j,l}) + (S_r \cdot w_{r,l}) + b_2 \tag{2.6}$$

Em seguida, o valor de E_l é enviado a função de ativação do neurônio. Portanto, a saída do neurônio l é dada por:

$$S_l = f(E_l) = \frac{1}{1 + e^{-E_l}} \tag{2.7}$$

Após o processamento em l, é feito o mesmo processamento em p e em v. Então, com todas as saídas dos neurônios da segunda camada oculta calculadas, o processamento passa para a próxima camada, sendo essa, a camada de saída.

Assim, os neurônios *0* e *1* começam seu processamento utilizando S_l, S_p e S_v como suas entradas. O neurônio *0* recebe em sua entrada os dados processados S_l, S_p e S_v e os pesos de suas arestas, respectivamente $w_{l,0}$ $w_{p,0}$ e $w_{v,0}$. Com isso, o neurônio *0* realiza a soma de suas entrada ponderadas com os seus pesos e acrescenta o valor do seu bias. Logo, a entrada de processamento do neurônio *0* é representada por:

$$E_0 = (S_l \cdot w_{l,0}) + (S_p \cdot w_{p,0}) + (S_v \cdot w_{v,0}) + b_3 \tag{2.8}$$

Em seguida, o valor de E_0 é enviado a função de ativação do neurônio. Portanto, a saída do neurônio *0* é dada por:

$$S_0 = f(E_0) = \frac{1}{1 + e^{-E_0}} \tag{2.9}$$

Após o processamento em *0*, é feito o mesmo processamento em *1*. Como os neurônios *0* e *1* fazem parte da camada de saída da rede, suas saídas são os resultados do processamento da rede. Logo, a execução da rede chega ao fim e as saídas obtidas como resultados são:

$$y_0 = S_0$$

$$y_1 = S_1$$

2.2.2 Aprendizagem

Para que a execução da rede neural consiga gerar os resultados esperados é necessário que os pesos da rede estejam configurados para a resolução do problema proposto. O processo de obtenção dos pesos ideais para resolução do problema é denominado período de aprendizagem da rede neural.

A aprendizagem da rede neural é composta principalmente pelo treinamento. Durante o treinamento, a rede é inicializada com pesos aleatórios e dados de entrada são utilizados para treinar a rede. O treinamento consiste em executar a rede neural várias vezes com dados próprios para o treinamento. A cada execução o resultado da rede é analisado e utilizado como métrica para correção e reajuste dos pesos, até que a rede se encontre pronta para utilização.

O treinamento de uma rede neural possui duas principais condições de parada: o erro mínimo alcançado e a quantidade de épocas executadas. Um conjunto de treinamento é composto por vários dados para treinamento, quando o erro produzido pela rede neural em todos os dados do treinamento é menor que um erro mínimo estipulado, conclui-se que a rede neural já aprendeu com aqueles dados e está pronta para uso, assim o treinamento termina por meio do erro mínimo alcançado. Uma execução completa de todos os dados de treinamento é chamada de época. A segunda maneira de terminar o treinamento é a execução de todos os dados de treinamento a quantidade de épocas estipulada.

2.2.2.1 Tipos de Treinamento

Existem três tipos de treinamento que são predominantemente utilizados em Redes Neurais:

- Treinamento Supervisionado
- Treinamento não Supervisionado
- Aprendizado por Reforço

No Treinamento Supervisionado os dados utilizados durante o treinamento contém a resposta da rede neural. Logo, a cada execução do treinamento a resposta gerada pela rede é comparada com a resposta esperada como correta e um erro é calculado. Esse erro é a base para o cálculo do reajuste dos pesos da rede. O presente trabalho utiliza um conhecido método de Treinamento Supervisionado chamado *backpropagation* para o treinamento da rede neural implementada.

No Treinamento não Supervisionado os dados utilizados para treinamento não contém a resposta que é esperada pela rede. Assim, não é possível calcular um erro a cada execução do treinamento, logo o treinamento consiste na capacidade da rede neural em conseguir agrupar semelhanças de suas entradas, criando grupos para se basear durante a execução.

Já no Aprendizado por Reforço existe uma recompensa ou punição por acerto ou erro da rede neural durante o treinamento. Por essa recompensa ou punição a rede neural consegue saber em qual direção evoluir. Porém, diferentemente do Treinamento Supervisionado a resposta correta de cada dado de treinamento nao é fornecida.

2.2.2.2 Backpropagation

O *Backpropagation* é um conhecido algoritmo de treinamento supervisionado. Tal algoritmo foi escolhido para o treinamento da rede neural implementada no presente trabalho. A ideia principal do *backpropagation* é calcular uma taxa de erro nos neurônios da camada de saída, durante cada execução de treinamento. Para o cálculo desse erro é utilizada a saída esperada da rede neural em cada execução, que é conhecida, visto que o treinamento é supervisionado. Contudo, as saídas esperadas nos neurônios intermediários da rede neural são desconhecidas. Por isso, o *backpropagation* retropropaga o erro da camada de saída para as camadas anteriores, calculando a influência de cada neurônio no erro final obtido. Após a retropropagação do erro até a camada de entrada, é feito o reajuste dos pesos e dos bias de cada neurônio. Assim, o *backpropagation* pode ser dividido em duas etapas: o *feedforward* e a retropropragação do erro.

Para o treinamento da rede neural com o metódo de *backpropagation*, o parâmetro taxa de aprendizagem deve ser definido. A taxa de aprendizagem, representada por η, é um indicador do quanto a rede neural deve se adaptar diante do erro final. Geralmente, η é um número entre 0 e 1, quanto mais próximo de 1 maior será a alteração dos pesos da rede a cada erro encontrado e vice versa.

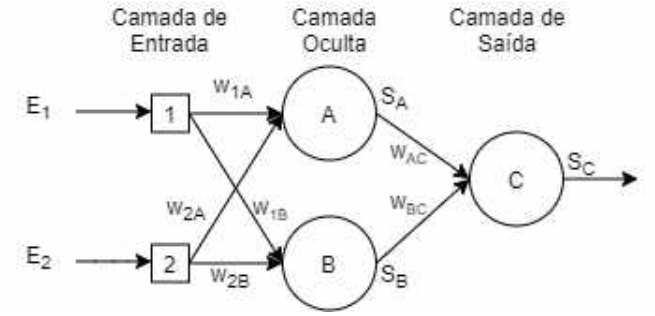


Figura 10 – Exemplo de Rede Neural

Fonte: Figura construída pelo autor.

A figura 10 retrata uma rede neural utilizada para exemplificar o algoritmo *backpropagation*. Em primeiro lugar, considerando E_1 e E_2 as entradas do conjunto de treinamento atual e $S_{esperado}$ a saída esperada da rede neural, é realizado o *feedforward*. O *feedforward* consiste em executar a rede neural, assim como descrito na subseção 2.2.1.3. Após a execução da rede, os dados processados S_A, S_B e S_C são conhecidos como as saídas dos neurônios A, B e C, respectivamente. Com isso, se inicia o cálculo do erro na camada de saída da rede. O erro é denotado por δ, logo o erro no neurônio C é dado por:

$$\delta_C = (S_{esperado} - S_C) \cdot f'(S_C) \tag{2.10}$$

Na equação 2.10, o termo $(S_{esperado} - S_C)$ é a diferença entre a resposta esperada

como correta e a resposta obtida pela rede neural. Já o termo $f'(S_C)$ é a derivada da função de ativação utilizada no neurônio. A derivada representa a tangente no ponto S_C da curva da função de ativação. A tangente, por sua vez, indica a inclinação da reta e logo é responsável por induzir a direção em que a rede neural deve corrigir seus pesos para diminuir o erro. Como a função de ativação utilizada é a sigmóide, sua derivada é:

$$f'(x) = f(x) \cdot (1 - f(x)), sendo f(x) = \frac{1}{1 + e^{-x}} \tag{2.11}$$

Logo, da equação 2.10 e 2.11, conclui-se que:

$$\delta_C = (S_{esperado} - S_C) \cdot (\frac{1}{1 + e^{-S_C}}) \cdot (1 - \frac{1}{1 + e^{-S_C}}) \tag{2.12}$$

Após calcular o erro em todos os neurônio da camada de saída, tem inicio o processo de retropropagação do erro. É calculado o erro em cada neurônio da última camada oculta, depois da penúltima, até a camada de entrada ser atingida. No exemplo da figura 10 o próximo neurônio a ter seu erro calculado é o neurônio A.

O erro em neurônios de camadas ocultas é calculado por:

$$\delta_A = \sum_{i=0}^{n} (\delta_i \cdot w_{Ai}) \cdot f'(S_A) \tag{2.13}$$

Sendo i cada neurônio presente na camada subsequente a que está sendo calculada, com os quais o neurônio A tem ligações através das arestas w_{Ai}. Assim, o erro no neurônio A do exemplo da figura 10 é dado por:

$$\delta_A = (\delta_C \cdot w_{AC}) \cdot (\frac{1}{1 + e^{-S_A}}) \cdot (1 - \frac{1}{1 + e^{-S_A}}) \tag{2.14}$$

O cálculo do erro no neurônio B é obtido de maneira idêntica ao cálculo do neurônio A. Assim que os erros em todos os neurônios foram calculados, é feito o reajuste dos pesos e do bias de cada neurônio. O reajuste é representado por:

$$w_{ij_{novo}} = w_{ij_{antigo}} + (\eta \cdot \delta_j \cdot S_i) \tag{2.15}$$

$$bias_{j_{novo}} = bias_{j_{antigo}} + (\eta \cdot \delta_j) \tag{2.16}$$

Nas equações 2.15 e 2.16, η indica a taxa de aprendizagem utilizada durante o treinamento, δ_j indica o erro calculado para o neurônio j e S_i indica a saída no neurônio i.

Portanto, no exemplo da figura 10, as equações de reajuste de pesos e bias para o neurônio A são:

$$w_{AC_{novo}} = w_{AC_{antigo}} + (\eta \cdot \delta_C \cdot S_A) \tag{2.17}$$

$$bias_{A_{novo}} = bias_{A_{antigo}} + (\eta \cdot \delta_A) \tag{2.18}$$

O reajuste dos pesos e bias é a última etapa do algoritmo *backpropagation*. Após isso a rede neural é executada com o próximo conjunto de treinamento e o algoritmo executado novamente. Esse processo acontece durante todo o treinamento, até que alguma das condições de parada do treinamento sejam alcançadas. No apêndice do trabalho há um pseudocódigo do algoritmo backpropagation.

3 Metodologia

A proposta do presente trabalho é utilizar técnicas de inteligência artificial que permitam encontrar um conjunto de características para predizer proteínas exportadas em bactérias patôgenicas. A metodologia é dividida em três etapas:

- Criação de uma base de proteínas para treinamento de uma rede neural.
- Criação, treinamento e teste de uma rede neural.
- Experimentos e avaliação dos resultados da rede neural comparados com o software Weka.

3.1 Criação de uma base de proteínas

A primeira parte da execução do trabalho foi a criação de uma base de proteínas para treinamento e testes da rede neural. Para isso, foram extraídas proteínas dos genomas de 35 espécies de bactérias patogênicas:

* Acinetobacter baumannii
* Acinetobacter calcoaceticus
* Acinetobacter lwoffii
* Bacteroides fragilis
* Campylobacter coli
* Campylobacter jejuni
* Clostridioides difficile
* Clostridium perfringens
* Clostridium tetani
* Corynebacterium striatum
* Enterobacteriaceae bacterium 9 2 54FAA
* Enterobacteriaceae bacterium ATCC 29904
* Enterobacteriaceae bacterium strain FGI 57

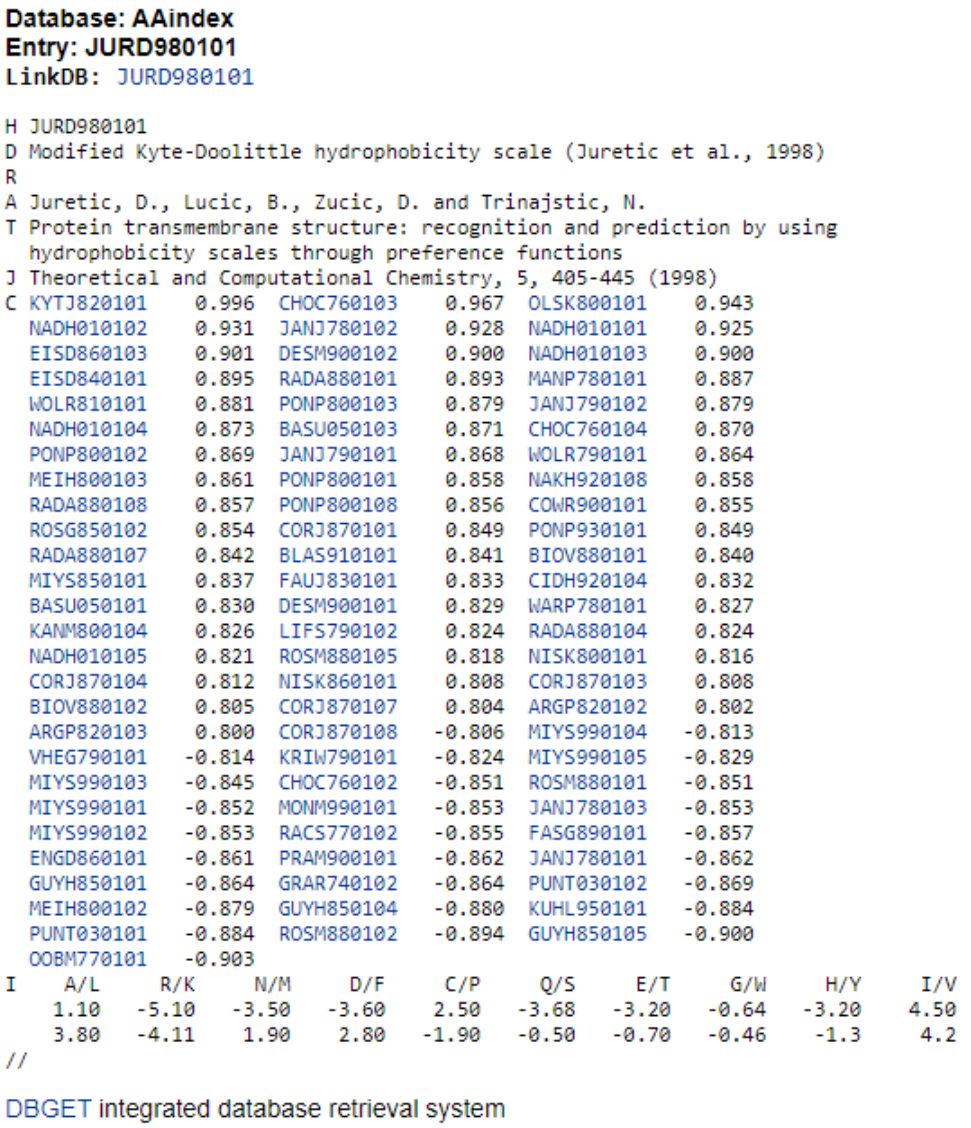
* Enterococcus avium
* Enterococcus faecalis
* Escherichia coli IAI39
* Escherichia coli O104 H4
* Escherichia coli O157 H7
* Escherichia coli O83 H1
* Escherichia coli str K-12
* Escherichia coli UMN026
* Klebsiella pneumoniae
* Mycobacterium tuberculosis
* Neisseria gonorrhoeae
* Pseudomonas aeruginosa
* Salmonella enterica Typhi CT18
* Salmonella enterica Typhimurium LT2
* Shigella dysenteriae
* Shigella flexneri
* Staphylococcus aureus
* Stenotrophomonas maltophilia
* Streptococcus agalactiae
* Streptococcus pneumoniae
* Streptococcus pyogenes
* Vibrio cholerae

A representação de cada proteína é formada por uma string de caracteres. Cada caracter indica um aminoácido da cadeia da proteína. Após a coleta das proteínas para base de dados foi feita uma busca de prováveis descritores/características sinalizadoras de exportação em proteínas. Para isso, informações foram extraídas de artigos indicando possíveis características químicas ou físicas sinalizadoras de exportação. A título de exemplo:

a proporção de uma proteína com estrutura tridimensional alfa-hélice, características físico-químicas diversas como hidrofobicidade, hidrofilicidade, quantidade de aminoácidos carregados negativamente ou positivamente e tipos de metais presentes em sítios catalíticos (BACHEM, 2020).

Para geração de descritores serão utilizados quatro grupos de classificação de aminoácidos conforme já demonstrado e explicado na figura 1. Os aminoácidos são divididos em aminoácidos de característica básica, não polar (hidrofóbicos), polar (hidrofílicas) e ácidas. Com esse conhecimento foi criada uma nova característica para o treinamento da rede levando em consideração a quantidade de aminoácidos com a mesma característica química (polar, não polar, básica, ácida).

Considerando experimentos prévios, sabe-se que apenas esses quatro descritores não são suficientes para classificar as proteínas. Por esse motivo, foi feita uma busca por novos descritores no repositório *AAindex1* (KAWASHIMA; OGATA; KANEHISA, 1999). Após tal pesquisa quatro preditores foram escolhidos para utilização: PARJ860101, JOND750101, EISD840101 e JURD980101. Tais preditores foram escolhidos por indicarem características hidrofóbicas na proteína. Na figura 11 é possível observar o preditor JURD980101 encontrado no repositório AAIndex.



Database: AAindex
Entry: JURD980101
LinkDB: JURD980101

```
H JURD980101
D Modified Kyte-Doolittle hydrophobicity scale (Juretic et al., 1998)
R
A Juretic, D., Lucic, B., Zucic, D. and Trinajstic, N.
T Protein transmembrane structure: recognition and prediction by using
  hydrophobicity scales through preference functions
J Theoretical and Computational Chemistry, 5, 405-445 (1998)
C KYTJ820101    0.996  CHOC760103    0.967  OLSK800101    0.943
  NADH010102    0.931  JANJ780102    0.928  NADH010101    0.925
  EISD860103    0.901  DESM900102    0.900  NADH010103    0.900
  EISD840101    0.895  RADA880101    0.893  MANP780101    0.887
  WOLR810101    0.881  PONP800103    0.879  JANJ790102    0.879
  NADH010104    0.873  BASU050103    0.871  CHOC760104    0.870
  PONP800102    0.869  JANJ790101    0.868  WOLR790101    0.864
  MEIH800103    0.861  PONP800101    0.858  NAKH920108    0.858
  RADA880108    0.857  PONP800108    0.856  COWR900101    0.855
  ROSG850102    0.854  CORJ870101    0.849  PONP930101    0.849
  RADA880107    0.842  BLAS910101    0.841  BIOV880101    0.840
  MIYS850101    0.837  FAUJ830101    0.833  CIDH920104    0.832
  BASU050101    0.830  DESM900101    0.829  WARP780101    0.827
  KANM800104    0.826  LIFS790102    0.824  RADA880104    0.824
  NADH010105    0.821  ROSM880105    0.818  NISK800101    0.816
  CORJ870104    0.812  NISK860101    0.808  CORJ870103    0.808
  BIOV880102    0.805  CORJ870107    0.804  ARGP820102    0.802
  ARGP820103    0.800  CORJ870108   -0.806  MIYS990104   -0.813
  VHEG790101   -0.814  KRIW790101   -0.824  MIYS990105   -0.829
  MIYS990103   -0.845  CHOC760102   -0.851  ROSM880101   -0.851
  MIYS990101   -0.852  MONM990101   -0.853  JANJ780103   -0.853
  MIYS990102   -0.853  RACS770102   -0.855  FASG890101   -0.857
  ENGD860101   -0.861  PRAM900101   -0.862  JANJ780101   -0.862
  GUYH850101   -0.864  GRAR740102   -0.864  PUNT030102   -0.869
  MEIH800102   -0.879  GUYH850104   -0.880  KUHL950101   -0.884
  PUNT030101   -0.884  ROSM880102   -0.894  GUYH850105   -0.900
  OOBM770101   -0.903
I    A/L     R/K     N/M     D/F     C/P     Q/S     E/T     G/W     H/Y     I/V
    1.10   -5.10   -3.50   -3.60    2.50   -3.68   -3.20   -0.64   -3.20    4.50
    3.80   -4.11    1.90    2.80   -1.90   -0.50   -0.70   -0.46    -1.3     4.2
//
```

DBGET integrated database retrieval system

Figura 11 – Vetor de pesos preditores de JURD980101

Fonte: Captura de tela encontrada em https://www.genome.jp/dbget-bin/www_bget?aaindex:JURD980101

As últimas duas linhas da tabela mostrada na figura 11 são os dados utilizados como pesos para representação das proteínas. Percebe-se que cada aminoácido tem um valor diferente de peso atrelado a ele. Assim, é construída uma tabela de pesos, proveniente das caracteristicas fisico-químicas e dos dados retirados dos preditores do AAIndex de cada aminoácido. A tabela 1 indica os pesos utilizados para criação dos dados de proteínas desse trabalho.

Referência	A	R	N	D	C	Q	E	G	H	I
Base	0	1	0	0	0	0	0	0	1	0
Ácido	0	0	0	1	0	0	1	0	0	0
Polar	0	0	1	0	1	1	0	1	0	0
Não Polar	1	0	0	0	0	0	0	0	0	1
Massa	89.09	174.20	132.12	133.10	121.16	146.15	147.13	74.07	155.16	131.18
Massa Molar	71.08	156.19	114.10	115.09	103.14	128.13	129.11	57.05	137.14	113.16
PARJ860101	2.10	4.20	7.0	10.0	1.4	6.0	7.8	5.7	2.1	-8.0
JOND750101	0.87	0.85	0.09	0.66	1.52	0	0.67	0.1	0.87	3.15
EISD840101	0.25	-1.76	-0.64	-0.72	0.04	-0.69	-0.62	0.16	-0.4	0.73
JURD980101	1.10	-5.10	-3.5	-3.6	2.5	-3.68	-3.20	-0.64	-3.20	4.5

Referência	L	K	M	F	P	S	T	W	Y	V
Base	0	1	0	0	0	0	0	0	0	0
Ácido	0	0	0	0	0	0	0	0	0	0
Polar	0	0	0	0	0	1	1	0	1	0
Não Polar	1	0	1	1	1	0	0	1	0	1
Massa	131.18	147.19	149.21	165.19	115.13	105.09	119.12	204.23	181.19	117.15
Massa Molar	113.16	128.17	131.20	147.18	97.12	87.08	101.10	186.21	163.17	99.13
PARJ860101	-9.20	5.7	-4.2	-9.2	2.1	6.5	5.2	-10	-1.9	-3.7
JOND750101	2.17	1.64	1.67	2.87	2.77	0.07	0.07	3.77	2.67	1.87
EISD840101	0.53	-1.1	0.26	0.61	-0.07	-0.26	-0.18	0.37	0.02	0.54
JURD980101	3.8	-4.11	1.9	2.8	-1.9	-0.5	-0.7	-0.46	-1.3	4.2

Tabela 1 – Indíce de propensão de pesos

Na tabela 1 cada aminoácido é representado por uma letra. Para cada letra existem dez pesos, indicados por uma característica fisico-química específica de cada aminoácido ou por um preditor do repositório AAIndex. As características dos aminoácidos utilizadas são: classificação (base, ácido, polar, não polar), massa e massa molar.

Com a tabela 1 criada, é feito um histograma para computar a frequência de cada aminoácido na proteína. Logo depois, a proteina é subdividida em três partes: início (primeiros 40 aminoácidos), meio (1/3 da proteína) e fim (últimos 60 aminoácidos). Com isso, é calculada novamente a frequência de cada aminoácido em cada grupo da proteína. No fim, quatro histogramas são gerados.

A partir dos histogramas gerados, é possível calcular o valor de cada referência da tabela 1 da proteína. Esse cálculo é representado na equação 3.1:

$$Referencia = \sum_{i=0}^{20} (fa_i \cdot w_{a_{ir}}) \tag{3.1}$$

Sendo:

· i: o aminoácido a ser calculado, variando na tabela de A até V.

· fa_i: a frequência do aminoácido i.

· $w_{a_{ir}}$: o peso do aminoácido i para a referência calculada.

Cada referência gera quatro dados indicadores da proteína. O primeiro calculado com a frequência dos aminoácidos em toda proteína, o segundo calculado com a frequência dos aminoácidos iniciais da proteína, o terceiro com a frequência dos aminoácidos finais da proteína e o último com a frequência dos aminoácidos no meio da proteína.

Portanto, uma proteína é representada na entrada da rede neural por um vetor de 60 números, na seguinte ordem:

- 20 indicadores da frequência de cada aminoácido na proteína toda, seguindo a disposição: fa_A, fa_R, fa_N, fa_D, fa_C, fa_Q, fa_E, fa_G, fa_H, fa_I, fa_L, fa_K, fa_M, fa_F, fa_P, fa_S, fa_T, fa_W, fa_Y, fa_V.
- 10 indicadores de referência utilizando a frequência dos aminoácidos em toda a proteína. As referências são organizadas na seguinte disposição: *Base*, *Ácido*, *Polar*, *Não Polar*, *Massa*, *Massa Molar*, *PARJ860101*, *JOND750101*, *EISD840101*, *JURD980101*.
- 10 indicadores de referência utilizando a frequência dos aminoácidos iniciais da proteína, na mesma ordem apresentada no item anterior.
- 10 indicadores de referência utilizando a frequência dos aminoácidos finais da proteína, na mesma ordem apresentada anteriormente.
- 10 indicadores de referência utilizando a frequência dos aminoácidos no meio da proteína, na mesma ordem apresentada anteriormente.

```
M K F K K L V S L G V L S T L L L S V I T G C S T D S S G S D S K S D S K T
K G D L T V Y T A I E E D S I E P Y L A T F K E K Y P D I K L N I V R A S T
G D I T A R L L A E K D N P Q A D V V W G V A A T S L L V A D D Q G M L E P
Y A P K G S E E I E S A F K D D K E V P S W V G I D A W M T G I T V N T K E
L S D K N I P V P S S Y E D L I K P E Y K G L I S M P N P S S S G T G Y L T
V S A L I Q I M G E E K A W Q Y M D K L H E N I G V Y T Q S G S A P A T S A
A S G E Y P I G I S F G Y R G I K L K E E G Y P V E V V F P K E G S G W D I
E A N A L V K K D N I K E A S K V F L D W A I S K D A M N E Y S K N Y A V T
T I S T G N P I P E G F P K K P L E Q M I D N D L K S A A K N R E D I L N K
W I S K Y D G K T E K E S

26 4 14 25 1 6 29 26 1 26 26 36 8 7 19 37 21 7 15 21 41.0 54.0 120.0 140.0
45181.27 38785.43 706.7 461.40994 -46.81 -164.20001 6.0 3.0 19.0 12.0 4809.37
4088.7197 86.49999 41.04 -5.000001 1.2799997 10.0 10.0 21.0 19.0 7895.5596
6814.55 168.0 77.78999 -16.14 -69.950005 9.0 17.0 46.0 47.0 14973.181 12829.22
236.89998 153.21 -9.6 -45.04001
```

Figura 12 – Exemplo de representação de proteína

Fonte: Figura construída pelo autor.

A figura 12 mostra uma proteína antes e depois do processamento para criação da base de dados. Na primeira parte da figura, tem-se uma sequência de caracteres, onde cada caracter indica um aminoácido na cadeia da proteína. Na segunda parte da figura são exibidos os 60 parâmetros numéricos finais que representam a proteína após o seu processamento.

Desse modo, foi construída uma base com 2500 proteínas, representadas cada uma da maneira detalhada anteriormente, utilizadas para treinamento e testes da rede neural.

3.2 Criação, treinamento e teste da Rede Neural

O próximo passo no desenvolvimento desse trabalho foi a implementação de uma rede neural utilizando a linguagem C. Essa rede foi implementada de um modo genérico, no qual o usuário consegue definir em tempo de execução vários parâmetros da rede. Os parâmetros que podem ser definidos pelo usuário são: a quantidade de neurônios na camada de entrada, a quantidade de camadas ocultas, a quantidade de neurônios em cada camada oculta, a quantidade de neurônios na saída, o erro mínimo para finalizar o treinamento, a quantidade de épocas para treinamento e a taxa de aprendizagem. A função de ativação por padrão é a função sigmóide, não podendo esta ser alterada pelo usuário.

A rede criada utiliza o algoritmo *backpropagation* para treinamento, com inicialização dos pesos como números aleatórios entre 0 e 1. O treinamento é feito através de um arquivo que indica a quantidade de dados para treinamento seguido por cada dado de entrada com sua respectiva saída correta. Após a finalização do treinamento, por quantidade de épocas máxima atingidas ou por erro mínimo encontrado, a rede pode ser testada lendo entradas de um arquivo e gerando um arquivo de saída com as respostas das entradas.

A rede neural desenvolvida está disponivel em https://github.com/tiagohugovitor/Rede-Neural-TCC. Tal ambiente possui exemplos dos arquivos utilizados para treinamento e teste da Rede, além de conter alguns programas utilizados para facilitar os experimentos e também os dados que foram gerados para experimentos e testes.

Para teste do funcionamento da rede neural foi utilizado um conhecido problema de classificação de flores Íris. O problema consiste em classificar flores do gênero Íris em três espécies diferentes. Os dados de treinamento utilizados para esse problema foram retirados da biblioteca Weka (WEKA, 2020). São 150 dados que consistem em quatro atributos de entrada e três possíveis espécies para classificação.

Para utilização dos dados na rede neural, foi feito antes uma normalização. A normalização consiste em um mapeamento de cada atributo dos dados do seu valor real para um valor correspondente entre 0 e 1, sendo que o maior valor do atributo corresponde

a 1 e, consequentemente, o menor valor do atributo corresponde ao 0. A normalização é necessária para que os dados de todos os atributos estejam na mesma escala e possam ser comparados, visto que, o processamento nos neurônios envolve dados de todos os atributos de entrada. A normalização do dado foi feita através da equação 3.2:

$$X_{normalizado} = \frac{(x - min)}{max - min} \tag{3.2}$$

Na equação 3.2, $X_{normalizado}$ indica o novo valor do atributo convertido, x indica o valor antigo do atributo, max representa o maior valor daquele atributo no conjunto de dados e min o menor valor do atributo naquele conjunto de dados.

O algoritmo de normalização utilizado está disponível para visualização em https://github.com/t Neural-TCC.

Foram construídos vários modelos diferentes da rede neural para executar o problema da classificação das flores Íris. Em todas as execuções a rede possuia quatro neurônios na camada de entrada, uma camada oculta, três neurônios na camada de saída e 0.001 de erro mínimo para fim do treinamento. Como existem três neurônios na camada de saída, espera-se que o primeiro neurônio receba um valor alto e os outros dois um valor baixo para indicar a primeira classe, o segundo neurônio receba um valor alto e os outros dois um valor baixo para indicar a segunda classe e assim por diante. Na tabela 2 são demonstradas as execuções feitas e seus respectivos resultados.

Execuções	Neurônios na C. Oculta	η	Épocas	T. de treinamento (ms)	T. de execução (ms)	Tx de acerto (%)
Execução 1	3	0.3	500	59	14	96.6
Execução 2	3	0.3	1000	92	24	97.3
Execução 3	3	0.5	500	49	23	97.3
Execução 4	3	0.5	1000	94	9	97.3
Execução 5	6	0.3	500	80	22	96.6
Execução 6	6	0.3	1000	149	23	97.3
Execução 7	6	0.5	500	79	24	97.3
Execução 8	6	0.5	1000	143	17	97.3

Tabela 2 – Execuções da Rede Neural construída para o problema Íris

Os resultados das execuções da rede neural foram satisfatórios para demonstração e aprovação de sua funcionalidade. É possível observar uma coerência nos resultados obtidos.

A rede tem sua melhor taxa de acerto em 97.3%. Assim, as únicas execuções que não conseguem convergir até a melhor taxa de acerto são a 1 e a 5. Tais execuções possuem configurações de taxa de aprendizado igual a 0.3 e 500 épocas para treinamento. Sendo o η um multiplicador que indica o quanto a rede deve aprender com seus erros, os resultados obtidos são explicados a seguir. Nas execuções 1 e 5 a rede não conseguiu convergir para a melhor resposta devido a sua baixa taxa de aprendizagem. Com o η igual a 0.3 os erros da rede eram corrigidos de maneira mais suave, logo o treinamento se torna mais demorado e somente 500 épocas não foram suficientes para a rede alcançar o seu máximo

de aprendizagem. Entretanto nas execuções com taxa de aprendizagem igual a 0.3 e 1000 épocas a rede já consegue atingir seu máximo devido ao maior número de épocas para o treinamento.

Nas execuções 3 e 7, a rede conseguiu atingir a melhor taxa de acerto mesmo com somente 500 épocas para treinamento. Tal feito é ligado ao alto valor da taxa de aprendizagem, 0.5. Utilizando 0.5 como η a convergência da rede neural até a melhor taxa de acerto acontece mais rapidamente, pois os erros encontrados em cada execução de treinamento são corrigidos com um maior peso de aprendizagem. Com isso, a rede das execuções 3 e 7 conseguem alcançar a taxa ótimo em apenas 500 épocas.

É importante ressaltar o lado negativo de se utilizar uma alta taxa de aprendizagem. Com uma alta taxa de aprendizagem, as redes podem não conseguir encontrar sua convergência para a melhor taxa de acerto. Isso acontece pois, durante o treinamento como o peso para correção do erro é alto, a rede oscila criando curvas ao redor da melhor convergência encontrada mas nunca a alcança. Felizmente, 0.5 não é uma taxa tão alta a ponto de desencadear esse evento de oscilação quando analisado o problema proposto.

Pode-se observar a indiferença nos resultados de 3 neurônios na camada oculta quando comparados aos de 6 neurônios na camada oculta. Assim, conclui-se que o problema não é tão dependente da quantidade de neurônios para seu processamento, visto que somente 3 neurônios conseguiram alcançar a melhor taxa de acerto. Portanto, o problema proposto é mais dependente da quantidade de épocas e da taxa de treinamento para conseguir fazer com que seus neurônios aprendam de uma forma apropriada e gerem a resposta desejada.

Com isso, tem-se como melhor execução o caso 3. Nesse, a melhor taxa de acerto de 97.3% é obtida com o menor número de recursos utilizados e logo menor tempo para treinamento da rede. Entende-se por recursos somente a quantidade de neurônios na camada oculta e o número de épocas para treinamento. A taxa de aprendizagem não é observada como recurso por sua alteração não resultar em diferenças no custo computacional da execução do algoritmo.

Para validação dos resultados obtidos foi utilizado o software Weka. O Weka é um software *open source* de aprendizado de máquina, amplamente utilizado para ensino, pesquisa e aplicações industriais (WEKA, 2020). Os resultados obtidos pela execução dos mesmos dados no software Weka são apresentados na tabela 3. Para execução no Weka foram utilizados os mesmos parâmetros da rede neural implementada.

Execuções	Neurônios na C. Oculta	η	Épocas	T. de treinamento (ms)	T. de execução (ms)	Tx de acerto (%)
Execução 1	3	0.3	500	60	0	98.6
Execução 2	3	0.3	1000	100	0	98.6
Execução 3	3	0.5	500	60	0	98.6
Execução 4	3	0.5	1000	110	0	98.6
Execução 5	6	0.3	500	80	0	98.6
Execução 6	6	0.3	1000	150	0	98.6
Execução 7	6	0.5	500	80	0	98.6
Execução 8	6	0.5	1000	160	0	98.6

Tabela 3 – Execuções do software Weka para o problema Iris

Existe uma constância no resultado das execuções do software Weka. Sua melhor taxa de acerto é 98.6%. As mudanças realizadas em todos os parâmetros não surgem efeitos na melhor taxa de acerto, que é encontrada com o mínimo de recursos dispostos, logo na primeira execução. Essa constância é esperada devido a maturidade do algoritmo Weka, com maior otimização no treinamento do que a rede implementada.

Ao comparar a melhor taxa de acerto da rede neural treinada (97.3%) com a taxa constante do Weka (98.6%) o resultado foi bem próximo e considerado satisfatório, provando que a rede neural implementada cumpre o seu papel.

É válido notar que os recursos utilizados para alcançar a melhor taxa em ambas as redes, implementada e do Weka, foram os mesmos: 3 neurônios na camada oculta e 500 épocas para o treinamento. Porém a rede implementada só conseguiu convergir para o ótimo utilizando taxa de aprendizagem de 0.5, enquanto o Weka conseguiu sua convergência para a melhor taxa de acerto já com η igual a 0.3.

Outro ponto importante de destaque na comparação entre a rede neural desenvolvida e as execuções no software Weka é o tempo de treinamento. A rede desenvolvida conseguiu realizar seu treinamento mais rapidamente do que o Weka em todas as execuções feitas, tendo tempo médio de treinamento de 93.125ms uma variância de 1136 e desvio padrão de 33.71. Já o tempo médio de treinamento do Weka é de 100ms, tendo variância de 1275 e desvio padrão de 35.7. Esse resultado é visto como ótimo, considerando que a rede, não possuindo nenhuma otimização, conseguiu praticamente se igualar e superar um pouco o Weka no tempo de treinamento. Tal feito pode ser explicado pela maior velocidade do C em ser executado quando comparado ao JAVA (linguagem de programação do WEKA). Por outro lado, a rede neural demora um maior tempo para realizar as execuções dos testes do que o software Weka.

4 Resultados e discussões

Nesse capítulo são demonstradas as execuções realizadas com a rede neural construída no caso de teste proposto de proteínas de bactérias patogênicas. A rede foi construída para classificar as proteínas de sua entrada em exportadas ou não exportadas. Portanto, a rede possui um único neurônio na camada de saída que emite 1.0 para indicar proteína exportada e 0.0 para indicar proteína não exportada.

Os dados de entrada da rede, utilizados para representar as proteínas, foram gerados conforme explicado na seção 3.1. Foram utilizados 1700 proteínas para treinamento e 800 proteínas para o teste da rede. Assim, a rede foi estruturada da seguinte maneira: 60 neurônios na camada de entrada, 1 neurônio na camada de saída e 0.001 de erro mínimo para fim do treinamento.

Os arquivos de treiamento e testes utilizados para obtenção dos resultados demonstados nessa seção podem ser encontrados em https://github.com/tiagohugovitor/Rede-Neural-TCC. Alguns programas auxiliares para normalização e correção dos testes também podem ser encontrados no mesmo link acima citado.

Antes da execução da rede os dados de entrada foram normalizados, assim como explicado na seção 3.2. Na tabela 4 são demonstradas as execuções feitas e seus respectivos resultados.

Execuções	C. Ocultas	Neurônios em cada C. Oculta	η	Épocas	T. de treinamento (ms)	T. de execução (ms)	Tx de acerto (%)
Execução 1	1	12	0.3	500	6502	58	72.8
Execução 2	1	12	0.3	1000	12472	40	79.6
Execução 3	1	12	0.5	500	6127	53	80.3
Execução 4	1	12	0.5	1000	11844	41	79.1
Execução 5	1	24	0.3	500	12186	45	76.1
Execução 6	1	24	0.3	1000	23896	63	80.1
Execução 7	1	24	0.5	500	11793	45	79.7
Execução 8	1	24	0.5	1000	23074	60	71.6
Execução 9	2	12	0.3	500	10482	41	82.1
Execução 10	2	12	0.3	1000	20395	41	79.3
Execução 11	2	12	0.5	500	10265	45	81.2
Execução 12	2	12	0.5	1000	20069	41	80.2

Tabela 4 – Execuções da rede neural construída para o problema de classificação de proteínas

A taxa de acerto da rede neural implementada para as execuções variou de 71.6% a 82.1%. É mais fácil realizar a análise dos resultados ao dividir as execuções em três grupos, conforme estrutura da rede. O primeiro grupo é composto pelas redes com 1 camada oculta de 12 neurônios, o segundo pelas redes com uma camada oculta de 24 neurônios e o terceiro pelas redes com 2 camadas ocultas de 12 neurônios cada.

Nos dois primeiros grupos acontece um crescimento na taxa de acerto ao aumentar o número de épocas mantendo o η igual a 0.3. Isso acontece pois possuindo mais épocas

para realizar o treinamento com uma taxa de aprendizagem baixa é esperado que a rede consiga convergir até chegar mais próxima da melhor taxa de acerto encontrada.

No primeiro e terceiro grupo acontece um ganho no acerto, ao trocar a taxa de aprendizagem de 0.3 para 0.5 e diminuir o número de épocas de 1000 para 500. Entretanto, no primeiro esse ganho supera o resultado da configuração de η igual a 0.3 e 500 épocas, enquanto no terceiro grupo isso não acontece. Tais diferenças podem levar a conclusões sobre as estruturas utilizadas.

Para a primeira estrutura, a melhor taxa de acerto é encontrada com uma alta taxa de aprendizagem e um baixo número de épocas. Isso ocorre, pois com uma única camada de poucos neurônios a rede tende a aprender mais com um maior número de épocas e a taxa de aprendizagem alta eleva a rapidez de sua aprendizagem. Contudo o aumento de ambas taxa de aprendizagem e número de épocas leva a um problema que acontece em todas as estruuras e será discutido posteriormente.

Já a segunda estrutura de uma camada oculta com 24 neurônios possui sua melhor taxa de acerto tendo 1000 épocas para treinamento com um η de 0.3. Isso acontece pois por possuir mais neurônios a curva criada pela rede consegue se adaptar mais do que a do primeiro grupo, portanto a taxa de 0.3 consegue chegar mais próximo da melhor curva encontrada ao ter um alto número de épocas para treinamento.

Com a terceira estrutura de duas camadas ocultas com 12 neurônios cada, a melhor taxa de acerto é alcançada com η igual a 0.3 e 500 épocas para treinamento. Sendo essa a configuração com a maior taxa de acerto entre todas as execuções.

Em todos os grupos ocorre uma queda na taxa de acerto ao aumentar o número de épocas com o η constante de 0.5. Essa queda pode ser explicada por duas razões: *overfitting* e alta taxa de aprendizagem. O overfitting acontece quando a rede é treinada por uma quantidade alta de épocas e assim se adapta exageradamente ao conjunto de treinamento, criando regras de padrões não somente genéricas mas também especificas daquele conjunto de treinamento. O problema relacionado a alta taxa de aprendizagem é a alta oscilação do resultado ao redor da melhor curva encontrada, mas nunca sua convergência para ela. Tal problema é descrito com mais detalhes na análise dos resultados do teste da rede neural para o problema Íris na seção 3.2.

É importante dar atenção a uma caracteristica do problema proposta em se adpatar melhor nas redes com duas camadas ocultas. Assim, diferentemente do exemplo Íris, duas camadas ocultas obtiveram desempenho melhor do que as de apenas uma camada. Com isso, a arquitetura com melhor taxa de acerto foi duas camadas oculta com 12 neurônios em cada, taxa de aprendizagem de 0.3 e 500 épocas para treinamento.

Os mesmos dados foram utilizados para execução no software Weka, os resultados podem ser observados na tabela 5.

Execuções	C. Ocultas	Neurônios em cada C. Oculta	η	Épocas	T. de treinamento (ms)	T. de execução (ms)	Tx de acerto (%)
Execução 1	1	12	0.3	500	7110	20	86.5
Execução 2	1	12	0.3	1000	14200	10	84.5
Execução 3	1	12	0.5	500	7130	10	86.5
Execução 4	1	12	0.5	1000	14240	20	86
Execução 5	1	24	0.3	500	14140	10	86
Execução 6	1	24	0.3	1000	28370	10	85.1
Execução 7	1	24	0.5	500	14170	10	87.2
Execução 8	1	24	0.5	1000	28500	10	83.2
Execução 9	2	12	0.3	500	8150	10	87.1
Execução 10	2	12	0.3	1000	16330	10	85.3
Execução 11	2	12	0.5	500	8160	10	85.7
Execução 12	2	12	0.5	1000	16200	10	84.8

Tabela 5 – Execuções do software Weka para o problema de classificação de proteínas

A taxa de acerto das execuções no software Weka variam de 83.2% a 87.2%. O melhor resultado é obtido na rede de uma camada oculta com 24 neurônios, η igual a 0.5 e 500 épocas. É interessante analisar que a tendência das taxas de acerto foram diminuir a medida que o número de epócas de treinamento aumentava, mantendo a taxa de aprendizagem constante. Tal feito pode ser explicado pelo *overfitting*, alcançado precocemente no Weka devido a sua maturidade no algoritmo de treinamento.

Ao comparar os resultados do software Weka com a rede neural implementada, observa-se que a estrutura da melhor taxa de acerto na rede neural é a estrutura da segunda melhor taxa de acerto do Weka. Isso indica um padrão, permitindo uma correlação entre os resultados do Weka e da rede implementada. Em tal correlação, a execução 8 pode ser apontada como a pior taxa de acerto e a execução 3 pode ser indicada como uma das melhores taxas de acerto. No geral, a rede teve um desempenho bom ao chegar próximo dos resultados do Weka, sendo o melhor resultado da rede implementada 82.1% e o do Weka 87.2%.

Em relação aos tempos de treinamento e execução, a rede implementada conseguiu superar o tempo de treinamento do Weka em todas as execuções com uma camada oculta. Nas execuções da estrutura com duas camadas ocultas, o Weka tem o tempo de treinamento melhor do que a rede implementada. Da mesma maneira, para realização dos testes, a rede ainda deixa a desejar, não conseguindo se igualar ao resultado constante do Weka de 10 ms em nenhuma execução. A rede implementada possui tempo de treinamento médio de 14092 com variância 34797550 e desvio padrão 5898, enquanto o Weka possui tempo de treinamento médio 14725 com variância de 48654290 e desvio padrão 6975.

Após as execuções dos dados no Weka utilizando Redes Neurais Artificiais, foram feitos testes para análise dos resultados sob a perspectiva de um outro algoritmo de Inteligência Artificial, as florestas aleatórias. O algoritmo de florestas aleatórias se baseia na criação de diversas árvores de decisão para o problema através dos dados de treinamento. Desse modo, em um teste todas as árvores são executadas e o resultado mais frequente nas árvores é o escolhido como resposta.

Execuções	Qtd. de árvores	Qtd. de features	T. de construção (ms)	T. de execução (ms)	Tx de acerto (%)
Execução 1	50	3	180	10	86.3
Execução 2	50	6	1910	10	86.1
Execução 3	50	12	400	10	86
Execução 4	100	3	280	20	86.7
Execução 5	100	6	450	30	87.1
Execução 6	100	12	800	20	86.5
Execução 7	200	3	550	60	87.6
Execução 8	200	6	900	40	87.2
Execução 9	200	12	1620	40	87.3

Tabela 6 – Execuções do software Weka utilizando florestas aleatórias para o problema de classificação de proteínas

Na tabela 6 são apresentados os resultados da execução do algoritmo de florestas aleatórias nos dados de classificação de proteínas. Entre as execuções foram feitas variações da quantidade de árvores utilizadas e da quantidade de features utilizadas para a construção de cada árvore. Observa-se que a taxa de acerto varia pouco entre 86% e 87.6%. Tais resultados são bem próximos dos alcançados pelo algoritmo de Redes Neurais do Weka. Quanto ao tempo de treinamento a média das execuções foram 787ms com variância 323661 e com desvio padrão 568.

O teste com o algoritmo de florestas aleatórias foi utilizado com o intuito de validar a aplicação de redes neurais na resolução do problema proposto. Com as execuções de florestas aleatórias, pode-se concluir que o emprego do algoritmo de redes neurais é válido para o problema proposto. Isso acontece devido a proximidade entre os resultados de ambos modelos, florestas aleatórias e redes neurais.

5 Considerações Finais

O objetivo principal do trabalho foi produzir uma rede neural capaz de classificar proteínas de bactérias patogênicas em exportadas ou não exportadas. Através da criação de uma base de dados de proteínas para treinamento e da implementação de uma rede neural com *backpropagation* na linguagem de programação C esse objetivo foi alcançado.

O trabalho atual propõe facilitar a localização de proteínas que sirvam para a produção de vacinas. As proteínas do tipo exportadas possuem grande chance de conter partes capazes de induzir respostas no sistema imunológico humano, causando a produção de anticorpos. Entretanto, como bactérias produzem inúmeras proteínas, é válido a construção de um sistema inteligente com o intuito de facilitar a localização de possiveis proteínas candidatas para a produção de vacinas.

A primeira parte da execução do trabalho foi criar uma base de dados de proteínas. Essa base de dados era composta por 2500 proteínas, representadas por um vetor de números indicando características que caracterizassem a proteína. Após isso, foi escolhido como sistema inteligente promissor para a classificação das proteínas: as Redes Neurais Artificiais. Assim, foi implementada uma rede neural em linguagem C, com utilização do algoritmo *backpropagation* para treinamento. Primeiramente, como teste, a rede neural foi utilizada para resolução do problema de classificação Íris. Os resultados foram promissores quando comparados com os do software Weka.

Para o problema de classificação Íris, a rede neural alcançou uma taxa de acerto de 97.3%, enquanto o Weka atingiu uma taxa de 98.6%. A rede implementada demorou em média 6.87 ms a menos para treinamento em relação ao tempo gasto pelo software Weka. Apresentado tais resultados, a rede foi dada como funcional, e foi direcionada para resolução do caso de teste proposto: a classificação de proteínas.

Para a classificação das proteínas foi utilizada a base de dados criada anteriormente para treinamento da rede neural. Os resultados encontrados apresentam que a rede conseguiu chegar perto da taxa de acerto do Weka, mas não alcança-lo. A rede implementada atingiu 82.1% de acerto, enquanto o Weka chegou a 87.2%. É válido ressaltar que a rede implementada realizou seu treinamento em um menor tempo que o Weka, considerando redes com uma camada oculta, isso se dá também pela rapidez da linguagem C em relação a linguagem de construção do Weka, Java.

Após a execução dos testes utilizando o modelo de Redes Neurias foi feita uma execução dos dados com o algoritmo de florestas aleatórias do Weka. A maior taxa de acerto obtida pelas florestas aleatórias foi 87.6%, valor bem próximo do resultado obtido pelas redes neurais. A proximidade dos resultados de ambos algoritmos foi utilizada como

reforço para demonstrar a aptidão do modelo de redes neurais para classificar e se adaptar ao problema proposto.

No geral, os resultados obtidos com o treinamento da rede através dos dados de proteínas criados foram considerados satisfatórios para a pesquisa, devido a sua proximidade do Weka. Era esperado que a rede implementada não conseguisse superar o Weka, visto a maturidade e as otimizações presentes no algoritmo do software.

Como trabalhos futuros, aponta-se melhorias que podem ser feitas no algoritmo implementado com o intuito de aumentar a taxa de acerto obtida e se aproximar ainda mais ou até ultrapassar os resultados do Weka. Tais melhorias podem ser alcançadas através da implementação de conceitos que evoluem o treinamento da rede, como o *momentum* e a validação cruzada. Além disso, destaca-se a possibilidade de aperfeiçoamento no método de representação de proteínas na base de dados. Essas podem ser aperfeiçoadas através de mudanças nas referências utilizadas com adição de novos preditores do AAIndex mais adequados ao problema ou até pela mudança geral na maneira de representar as proteínas.

6 Apêndice

Nesse Apêndice é detalhado o algoritmo backpropagation utilizado na construção da Rede Neural do presente trabalho.

1. Leitura dos dados para treinamento e armazenamento em um vetor.
2. Para cada época do treinamento, faça:
   2.1. Para cada dado de treinamento, faça:
      2.1.1. Cada neurônio da camada de entrada recebe os dados de entrada e propaga esses sinais para frente.
      2.1.2. Para cada camada oculta, faça:
         2.1.2.1. Cada neurônio da camada oculta recebe os dados da camada anterior, realiza a soma de suas entradas com os respectivos pesos e submete esse resultado a função de ativação.

$$Saída = f(\sum_{i=1}^{n}(a_i \cdot w_i) + bias) \quad (6.1)$$

      2.1.3. Para cada neurônio da camada de saída, faça:
         2.1.3.1. Cada neurônio da camada de saída recebe os dados da camada anterior, realiza a soma de suas entradas com os respectivos pesos e submete esse resultado a função de ativação.

$$Saída = f(\sum_{i=1}^{n}(a_i \cdot w_i) + bias) \quad (6.2)$$

      2.1.4. Cada neurônio de saída calcula seu Erro e seu delta:

$$Delta = (Saida_{Esperada} - Saida_{Recebida}) \cdot (1 - Saida_{Recebida}) \quad (6.3)$$

      2.1.5. Testar condição de parada por erro mínimo.
      2.1.6. Para cada camada oculta, começando da última e indo até a primeira, faça:
         2.1.6.1. Para cada neurônio n na camada oculta, calcule o delta do neurônio, utilizando o delta dos neurônios k da camada posterior.

$$Delta_n = \sum_{i=1}^{k}(delta_k \cdot w_{nk}) \cdot Saída_n \cdot (1 - Saída_n) \quad (6.4)$$

      2.1.7. Para cada camada oculta a partir da primeira, faça:

2.1.7.1. Para cada neurônio da camada oculta, atualize seu bias e seu vetor de pesos:

$$w_{ij_{new}} = w_{ij_{old}} + taxaAprendizado \cdot Delta_j \cdot Saida_i \tag{6.5}$$

$$bias_{i_{new}} = bias_{i_{old}} + taxaAprendizado \cdot Delta_i \tag{6.6}$$

2.1.8. Para cada neurônio da camada de saída atualize seu bias:

$$bias_{i_{new}} = bias_{i_{old}} + taxaAprendizado \cdot Delta_i \tag{6.7}$$

Referências

ALVES, S. M. A bioinformática e sua importância para a biologia molecular. *Revista Brasileira de Educação e Saúde*, v. 3, n. 4, p. 18–25, 2013. ISSN 2358-2391. Citado na página 10.

BACHEM. 2020. Disponível em: ⟨https://www.bachem.com/⟩. Citado na página 29.

BACKES, A. *UTILIZANDO O SOFTWARE WEKA*. 2020. Disponível em: ⟨http://www.facom.ufu.br/~backes/pgc204/Aula-UsandoWeka.pdf⟩. Nenhuma citação no texto.

BENDTSEN, J. D. et al. Non-classical protein secretion in bacteria. *BMC Microbiology*, v. 5, p. 1–13, 2005. ISSN 14712180. Citado na página 11.

COSTA, J. V.; SANTOS, A. Predição de proteínas de bactérias secretadas por via não clássicas. n. i, 2018. Nenhuma citação no texto.

FIGUEIROA, H. C. *Redes Neurais No Weka*. 2017. Disponível em: ⟨https://docplayer.com.br/40641925-Redes-neurais-no-weka.html⟩. Nenhuma citação no texto.

KAPP, K. et al. Post-Targeting Functions of Signal Peptides. 2009. Citado na página 13.

KAWASHIMA; OGATA, H.; KANEHISA, M. AAindex: Amino acid index database. *Nucleic Acids Research*, v. 27, p. 368—-369, 1999. Citado na página 29.

KHANACADEMY. *Direcionamento de proteínas*. 2019. Disponível em: ⟨https://pt.khanacademy.org/science/biology/gene-expression-central-dogma/translation-polypeptides/a/protein-targeting-and-traffic⟩. Citado 2 vezes nas páginas 10 e 14.

KHANACADEMY. *O neurônio, estrutura e função*. 2020. Disponível em: ⟨https://pt.khanacademy.org/science/6-ano/vida-e-evoluo-os-sistemas-do-corpo-humano/os-neuronios/a/o-neuronio-estrutura-e-funcao⟩. Citado na página 16.

LUNDEGAARD, C. et al. NetMHC-3.0: accurate web accessible predictions of human, mouse and monkey MHC class I affinities for peptides of length 8-11. *Nucleic acids research*, v. 36, n. Web Server issue, p. 509–512, 2008. ISSN 13624962. Nenhuma citação no texto.

NETO, A. d. M. Aprimoramento da anotação N-terminal de proteínas através da predição de peptídeo sinal em proteínas ortólogas e desenvolvimento de uma ferramenta automática para a identificação de grupos ortólogos contendo erros de anotação. p. 107, 2012. Nenhuma citação no texto.

REZENDE, A. d. F. S. et al. In silico identification of Corynebacterium pseudotuberculosis antigenic targets and application in immunodiagnosis. *Journal of Medical Microbiology*, v. 65, n. 6, p. 521–529, 2016. ISSN 00222615. Citado na página 10.

ROBERTIS, E. D.; HIB, J. *Biologia Celular e Molecular*. 4. ed. [S.l.: s.n.], 2006. Citado 2 vezes nas páginas 10 e 12.

RUSSELL, S.; NORVIG, P. *Inteligência Artificial*. 3. ed. [S.l.: s.n.], 2013. Citado 4 vezes nas páginas 10, 15, 16 e 17.

SANTOS, A. R. et al. Mature Epitope Density - A strategy for target selection based on immunoinformatics and exported prokaryotic proteins. *BMC Genomics*, BioMed Central Ltd, v. 14, n. Suppl 6, p. S4, 2013. ISSN 14712164. Disponível em: ⟨http://www.biomedcentral.com/1471-2164/14/S6/S4⟩. Citado na página 10.

Vale de Morais, R. L.; OLIVEIRA, G. Análise computacional do genoma de Schistosoma mansoni para identificação de proteínas potencialmente imunogênicas Análise computacional do genoma de Schistosoma mansoni para identificação de proteínas potencialmente imunogênicas. 2012. Nenhuma citação no texto.

WEKA. *https://www.cs.waikato.ac.nz/ml/weka/*. 2020. Disponível em: ⟨https://www.cs.waikato.ac.nz/ml/weka/⟩. Citado 2 vezes nas páginas 33 e 35.

ZHANG, G. P. Neural Networks for Classification: A Survey. v. 30, p. 451–462, 2000. Citado na página 15.