FACULDADES INTEGRADAS DE CARATINGA

FACULDADE DE CIÊNCIA DA COMPUTAÇÃO

ANÁLISE DO ALGORITMO SAND PARA INICIALIZAÇÃO DE PESOS DE REDES NEURAIS ARTIFICIAIS

VINÍCIUS CAMPISTA BRUM

Caratinga
2012

Vinícius Campista Brum

ANÁLISE DO ALGORITMO SAND PARA INICIALIZAÇÃO DE PESOS DE REDES NEURAIS ARTIFICIAIS

Monografia apresentada à banca examinadora da Faculdade de Ciência da Computação das Faculdades Integradas de Caratinga, como requisito parcial para obtenção do título de bacharel em Ciência da Computação, sob orientação do professor Msc. Jacson Rodrigues Correia da Silva.

Caratinga

2012

Vinícius Campista Brum

ANÁLISE DO ALGORITMO SAND PARA INICIALIZAÇÃO DE PESOS DE REDES NEURAIS ARTIFICIAIS

Monografia submetida à Comissão examinadora designada pelo Curso de Graduação em Ciência da Computação como requisito para obtenção do grau de Bacharel.

_____
Prof. Msc. Jacson Rodrigues Correia da Silva
Faculdades Integradas de Caratinga

_____
Prof. Wanderson Miranda Nascimento
Faculdades Integradas de Caratinga

_____
Prof. Msc. Fabrícia Pires Souza Tiola
Faculdades Integradas de Caratinga

Caratinga, 06/12/2012

AGRADECIMENTOS

Em primeiro lugar agradeço a Deus, por ter me guiado durante esse período de conquistas, vitórias, dificuldades, entre outras diversas situações. Por ter me guiado durante toda minha vida.

A toda minha família, especialmente a meus pais, meus avós e meu irmão pela base, apoio, incentivo e compreensão constantes.

Agradeço a todos os mestres que com certeza contribuíram para minha formação acadêmica, profissional e pessoal. Agradeço especialmente aos mestres, orientadores, professores, e amigos: Jacson Rodrigues Correia da Silva, Glauber Luís da Silva Costa e Vagner Aquino Zeferino, pelo apoio e incentivo.

A todos os amigos e irmãos de turma com quem convivi durante esse período, compartilhando momentos felizes e momentos difíceis. Agradeço a todos os amigos conquistados durante esses anos.

“O coração do homem planeja o seu caminho, mas é Deus que firma os seus passos.”

Provérbios 16,9

RESUMO

Devido ao grande crescimento da Internet e ao fato de mais pessoas se conectarem e realizarem suas tarefas através da Internet a cada dia, a segurança da informação é uma grande preocupação da sociedade e dos especialistas da área. Devido a isso, existem diversas ferramentas utilizadas para a segurança das redes de computadores, como por exemplo, os Sistemas de Detecção de Intrusão. Estudos de diversos pesquisadores incentivam a aplicação de técnicas de Inteligência Artificial a essas ferramentas por apresentarem aumento de flexibilidade e taxa de acerto, fornecendo assim mais confiabilidade.

Considerando esses fatos, esse trabalho tem por objetivo analisar a utilização do conceito de diversidade no repertório de anticorpos do Sistema Imunológico para inicialização de pesos de Rede Neural Artificial, por meio do algoritmo SAND (*Simulated ANnealing approach for Diversity*) com intuito de aumentar a diversidade ou área de cobertura dos pesos inicias, buscando assim melhorias quanto a tempo de treinamento e taxa de erro da Rede Neural Artificial.

Para realização desse trabalho foram utilizadas configurações de redes neurais apresentadas em outro trabalho da área que apresentou configurações mais adequadas para reconhecimento de padrões de ataques dentro da base de dados NSL-KDD. A partir dessas configurações foram realizados testes de treinamento e execução com a rede neural original e com as redes neurais alteradas pelo algoritmo SAND, comparando os resultados obtidos quanto à tempo de treinamento e taxa de acerto.

Com os resultados obtidos foi possível chegar à conclusão que o sucesso da utilização do SAND depende de como as mutações são realizadas. Em determinadas situações o algoritmo atinge taxas excelentes de diversidade, porém apresenta baixas melhorias no treinamento e na execução da rede. Sendo necessário realizar testes com diferentes formas de mutação para encontrar a forma que gera melhores resultados.

Palavras-chave: sistemas inteligentes, rede neural artificial, inicialização de pesos, diversidade de repertório, SAND

ABSTRACT

Due to the large growth of the Internet and the fact more people connect and conduct their tasks through the Internet every day, information security is a major concern of society and experts of area. Due to this, there are various tools used for security of computer networks, eg the Intrusion Detection Systems. Studies of several researchers encourage the application of Artificial Intelligence techniques for these tools for they present increase of flexibility and accuracy rate, thus providing more reliability.

Considering these facts, this work aims to analyze the use of the concept of diversity in the repertoire of antibodies of the immune system to boot weights of Artificial Neural Network, using the SAND algorithm (Simulated Annealing approach for Diversity) in order to increase the diversity or coverage area of the initial weights, thus seeking improvements as to the training time and error rate of the Artificial Neural Network.

To carry out this work were used neural network configurations shown in other work area settings which presented settings more suitable for pattern recognition of attacks within the data base NSL-KDD. From these settings were performed tests of training and execution with original neural network and with the neural networks changed by the algorithm SAND, comparing the results obtained as to the training time and hit rate.

With the results obtained we arrive the conclusion that the results of the use of SAND depend on how the mutations are performed. In certain situations the algorithm achieves excellent rates of diversity, but it has poor results in the training and implementing of the network. Being necessary to conduct tests with different forms of mutation to find the form that generates the best results.

Keywords: intelligent systems, artificial neural network, weights initialization, diversity of repertoire, SAND

LISTA DE SIGLAS

IA – Inteligência Artificial
MSE – Mean Square Error
R2L – Remote to Local
RNA – Rede Neural Artificial
SDI – Sistema de Detecção de Intrusão
SIA – Sistema Imunológico Artificial
SIH – Sistemas Inteligentes Híbridos
U2R – User to Root

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE GRÁFICOS

SUMÁRIO

1 INTRODUÇÃO

Hoje em dia, tarefas que fazem parte da rotina, como compras, transações bancárias, dentre outras, podem ser realizadas facilmente através da Internet, o que facilita muito o dia a dia das pessoas e contribui ao grande crescimento da Internet (CERT.BR, 2012).

Esse grande crescimento da Internet e o grande avanço dos dispositivos móveis implicam em um aumento significativo no volume de dados confidenciais que trafegam pela Internet, como números de contas, senhas bancárias, entre outros, e na necessidade de manter a segurança dessas informações. Da mesma forma que o número de dados aumenta a cada dia, aumenta também o número de usuários com pouco conhecimento quanto a segurança e pessoas mal intencionadas, uma vez que a própria Internet, além de disponibilizar soluções para a sociedade, disponibiliza também ferramentas e manuais para invasão de redes de computadores. Devido a isso, existem diversas ferramentas utilizadas para a segurança de redes de computadores, como por exemplo, o Sistema de Detecção de Intrusão (SDI).

Estudos de diversos pesquisadores mostram que os SDIs apresentam problemas como incapacidade de adaptação a novas formas de ataque, pois exigem grande esforço na manutenção da base de dados de assinaturas e possuem baixa taxa de acerto, o que afeta diretamente em sua confiabilidade (BRITT, 2007; WU, 2010).

Para suprir essas necessidades, diversos pesquisadores encorajam a aplicação de técnicas de Inteligência Artificial (IA) a essas ferramentas, uma vez que essas técnicas apresentam grande capacidade de adaptação a novas formas de ataques, o que aumenta a flexibilidade da ferramenta e a taxa de acerto. Vários métodos de IA são propostos para essa tarefa, como Rede Neural Artificial (RNA) e Sistema Imunológico Artificial (SIA), pesquisadores incentivam também a utilização de Sistemas Inteligentes Híbridos (SIH), que são sistemas que utilizam dois ou mais métodos para a realização de determinada tarefa, nas quais pelo menos um desses métodos é um método de IA, uma vez que cada método possui vantagens e desvantagens (SILVA, 2011).

Dessa forma o que justifica a realização desse trabalho é a necessidade constante de melhorias em ferramentas de segurança computacional, devido ao grande crescimento da Internet e do número de dados e usuários, tanto na busca de novas técnicas, quanto na busca de melhorias para as técnicas já existentes.

O objetivo desse trabalho foi analisar a utilização do conceito de diversidade do Sistema Imunológico para inicialização de pesos de determinada RNA com intuito de aumentar a diversidade, ou seja, a área de cobertura de seus pesos iniciais, buscando assim melhorias quanto a tempo de treinamento e taxa de acerto. Dessa forma o objetivo é buscar melhorias para uma técnica já existente apresentada em Silva (2011), através do uso de um algoritmo para diversificação de pesos iniciais de RNA apresentado em De Castro (2001).

Para realização dessa tarefa, foi escolhido o algoritmo SAND (*Simulated ANnealing approach for Diversity*) que tem por objetivo principal gerar uma população de ampla cobertura do espaço de buscas, ou seja, uma população diversificada sem qualquer conhecimento prévio do problema ou dos dados que serão utilizados durante o treinamento e teste (DE CASTRO, 2001).

A escolha do algoritmo SAND se baseia no resultado apresentado em De Castro (2001), no qual o algoritmo se destaca apresentando melhor desempenho se comparado com outros métodos. Outro fator que contribuiu para a escolha do SAND é o fato do algoritmo utilizar um dos conceitos mais fundamentais do Sistema Imunológico e consequentemente do SIA, que é a diversidade de repertório e o fato que o SIA pode ser considerado como uma área recente da IA; devido a extrema adequação de metáforas do sistema imune no processo de detecção de intrusos, e a possibilidade de apresentar bons e novos resultados, apresentadas no trabalho de Uchôa (2009).

Para alcançar o objetivo do trabalho, inicialmente foram implementados o algoritmo SAND apresentado em De Castro (2001) e a rede neural apresentada em Silva (2011). Após as implementações foram realizados testes de treinamento e execução da rede neural original e das redes neurais alteradas pelo algoritmo SAND, comparando os resultados obtidos quanto à tempo de treinamento e taxa de acerto.

2 REFERENCIAL TEÓRICO

Nesse capítulo são apresentadas técnicas, ferramentas e conceitos utilizados no desenvolvimento do trabalho, basicamente estruturados em introdução e pontos mais relevantes ao trabalho. O primeiro conceito apresentado é segurança computacional e sua importância, logo em seguida são apresentados conceitos de SDI, quanto aos primeiros passos, etapas, categorias, vulnerabilidades e a utilização de técnicas de IA. Após a apresentação dos conceitos de segurança computacional e SDI, são apresentadas as técnicas de IA envolvidas nesse trabalho.

2.1 SEGURANÇA COMPUTACIONAL

2.1.1 Introdução

A segurança computacional é um processo contínuo que tem o objetivo de fornecer serviços e ferramentas que garantam a segurança das informações, permitindo que os usuários possam utilizar as soluções disponibilizadas sem colocar em risco seus interesses e dados confidenciais (SILVA, 2011).

O processo de segurança envolve quatro passos (SILVA, 2011):

1. avaliação: é o primeiro passo e pode ser considerado o mais importante. Consiste na avaliação e determinação de medidas que garantam a segurança da empresa ou organização;
2. proteção: consiste na aplicação das medidas definidas no passo de avaliação. Sendo controle de acesso, afinação de tráfego e *proxies*[1], exemplos importantes de itens do processo de proteção;

[1] *proxies*: servidores que atuam entre um cliente e um servidor. São utilizados normalmente para controlar acesso a serviços e a Internet (CERT.BR, 2012).

3. detecção: é o processo de identificação do intruso e de eventos ou comportamentos suspeitos. A detecção consiste de quatro passos, que são: a coleta, a identificação, a validação, e o escalonamento; e
4. resposta: consiste em tomar medidas contra a invasão.

Basicamente a segurança envolve três elementos (CERT.BR, 2012; SILVA, 2011):

1. integridade: proteger de modificações não autorizadas;
2. confiabilidade ou sigilo: proteger de acesso não autorizado; e
3. disponibilidade: disponibilizar a informação ou recurso sempre que necessário de acordo com a autorização.

Os serviços devem fornecer esses elementos e possuir três procedimentos (CERT.BR, 2012; SILVA, 2011):

1. autenticação: processo de confirmação de identidade, ou seja, a identificação do usuário;
2. autorização: trata da concessão de permissões e privilégios aos usuários previamente identificados; e
3. não-repúdio: trata de assinaturas para evitar que um usuário possa negar que realizou alguma ação.

2.1.2 Importância

A cada dia que passa cresce o número de tarefas do cotidiano realizadas através da Internet, e consequentemente o número de usuários, sendo que grande parte desses usuários se vêem totalmente dependentes dessas facilidades (CERT.BR, 2012). Diversas tarefas como transações bancárias, compras, relacionamentos, entre outras, podem ser realizadas sem que o cliente ou o usuário precise se deslocar até a agência, ou loja mais próxima, gerando facilidades e oportunidades na vida de muitas pessoas, e contribuindo com o crescimento da Internet.

Esse crescimento implica em um aumento significativo no volume de dados que trafegam pela Internet, e no aumento do número de pessoas mal intencionadas,

ou seja, da mesma forma que cresce o número de usuários, cresce o número de dados, e o número de pessoas mal intencionadas, uma vez que a Internet disponibiliza soluções para a sociedade, e disponibiliza também ferramentas e manuais para invasão de redes de computadores.

Dessa forma a cada dia que passa cresce a necessidade de melhorias em ferramentas de segurança computacional, tanto na busca de novas técnicas e ferramentas, quanto na busca de melhorias para as técnicas e ferramentas já existentes, como os SDIs.

2.2 SISTEMAS DE DETECÇÃO DE INTRUSÃO

Nessa seção são apresentados conceitos de SDI, que é a ferramenta que se busca melhorar com a aplicação de técnicas de IA, basicamente RNA e o algoritmo SAND. São apresentados a seguir tópicos como histórico, objetivos, etapas, categorias e vulnerabilidades.

2.2.1 Introdução

Os primeiros passos dos sistemas para detecção de invasões foram dados por James P. Anderson, Dorothy Denning e Peter Neumann na década de 1980, através de pesquisas que mostravam que a forma de agir do invasor e do usuário normal eram diferentes, e que essas diferenças poderiam ser medidas e analisadas através do uso de ferramentas que automatizavam o processo de detecção, chamadas Sistemas de Detecção de Intrusão (COSTA, 2007; WANG, 2009).

Sendo assim, os SDIs são ferramentas que têm o objetivo de procurar e identificar invasões passadas e em curso, e alertar ao administrador do sistema. Os SDIs podem trabalhar sobre duas abordagens quanto ao momento de detecção, uma abordagem preemptiva, na qual monitoram e agem assim que observam a

atividade suspeita; ou uma abordagem evolucionária, na qual se baseiam em logs do sistema, agindo com segundos de atraso (SILVA, 2011).

Basicamente os SDIs possuem três etapas comuns (SILVA, 2011):

1. avaliação: etapa na qual são determinadas as necessidades de segurança, produzindo assim um perfil de segurança;
2. detecção: etapa na qual são identificadas as anomalias, coletando e analisando informações do sistema.
3. alarme: etapa na qual são emitidos os alertas caso seja detectada alguma invasão.

Os SDIs basicamente são categorizados de acordo com sua visão ou arquitetura: rede, host ou distribuída; quanto a sua forma de detecção: assinatura, anomalia ou híbrida; quanto ao seu comportamento pós-detecção: passivo ou ativo; e quanto à frequência de uso: monitoramento contínuo ou análise periódica (MACHADO, 2005; SILVA, 2011).

2.2.2 Vulnerabilidades

Atualmente as técnicas utilizadas nos SDIs apresentam falhas como a baixa taxa de acerto, ou seja, grande número de alarmes falsos apresentados nos métodos de detecção por anomalia; e a incapacidade de auto adaptação, ou seja, a incapacidade de aprender e tratar novas formas de ataque apresentadas nos métodos de detecção por abuso.

Para resolver esses problemas, ou pelo menos reduzi-los, diversos autores sugerem a aplicação de IA sobre esses sistemas, com o intuito de melhorar a taxa de acerto e possibilitar o aprendizado, o que afeta na confiabilidade e flexibilidade da ferramenta.

2.2.3 Sistemas de detecção de intrusão com técnicas de Inteligência Artificial

A aplicação de técnicas de IA sobre os SDIs é utilizada com o objetivo de melhorar os sistemas atuais, resolvendo ou pelo menos reduzindo os problemas citados como a incapacidade de adaptação a novas formas de ataque e a baixa taxa de acerto.

Para essa tarefa são propostos vários métodos de IA, como Redes Neurais Artificiais, Lógica Nebulosa, Métodos Computacionais Evolutivos, utilizados no trabalho de Silva (2011), por exemplo; Inteligência Coletiva, Sistema Imunológico Artificial, utilizados nos trabalhos de Machado (2005) e de Uchôa (2009), por exemplo; entre diversos outros métodos.

Porém, é recomendado por diversos autores, o uso de mais de um método para essa tarefa, uma vez que cada método possui vantagens e desvantagens, ou seja, recomenda-se o uso de Sistemas Inteligentes Híbridos, que tem o objetivo de juntar vários métodos para que as vantagens apresentadas por alguns, atendam as desvantagens apresentadas por outros.

2.3 SISTEMA IMUNOLÓGICO ARTIFICIAL

Nessa seção são apresentados conceitos básicos de SIA, que é a técnica na qual se baseia o algoritmo para inicialização dos pesos da RNA, com o objetivo de contextualizar o leitor das definições, princípios e elementos utilizados para o desenvolvimento desse trabalho. São apresentados a seguir tópicos como histórico, objetivos, aplicações, princípios e elementos.

2.3.1 Introdução

Os Sistemas Imunológicos Artificiais surgiram na década de 1980 por meio do trabalho de Farmer, Packard e Perelson (1986), trabalho que é considerado como pioneiro da área, a partir do conceito do Sistema Imunológico Humano, que se pode definir basicamente como a defesa do organismo contra invasores, a principal barreira do hospedeiro ou organismo contra invasores, sendo necessário distinguir o que faz (*self*) e o que não faz (*non-self*) parte do organismo (DE CASTRO, 2001; MACHADO, 2005). Sendo assim, o SIA é uma técnica que tem o objetivo de simular conceitos do Sistema Imune, para resolução de problemas em Computação e diversas áreas do conhecimento, como reconhecimento de padrões, segurança computacional, robótica, aprendizagem de máquina, otimização, detecção de falhas e anomalias, análise de dados, entre outros (DE CASTRO, 2001; MACHADO, 2005; UCHÔA, 2009).

Apesar de surgir na década de 1980, o SIA só passou a ser considerado como uma área da IA na década de 1990, a partir de *workshops* e conferências realizadas inicialmente no Japão em 1996 que despertaram iniciativas de consolidação e integração do SIA como uma linha de pesquisa. Desde então, anualmente são organizadas sessões e conferências para tratar assuntos da área (DE CASTRO, 2001; UCHÔA, 2009).

A seguir são apresentados princípios e elementos utilizados nesse trabalho com o intuito de contextualizar o leitor quanto a termos utilizados mais a frente, sendo basicamente anticorpos e diversidade imunológica.

2.3.2 Princípios e elementos básicos

Como citado anteriormente na seção 2.3.1, o sistema imunológico é a principal barreira do hospedeiro contra invasores, que tem o objetivo de distinguir o *self* e o *non-self* do sistema ou organismo. O sistema imunológico possui dois tipos

de resposta que trabalham em conjunto, uma resposta mais rápida, efetuada pelo sistema imune inato; e uma resposta mais lenta, efetuada pelo sistema imune adaptativo (DE CASTRO, 2001).

O sistema imune inato contém células que estão imediatamente prontas para combater diversos tipos de invasores, não exigindo exposição prévia dessas células aos invasores, ou seja, é uma resposta imune genérica. Já o sistema imune adaptativo trata a produção de anticorpos, que são glicoproteínas, também conhecidos como imunoglobulina, capazes de neutralizar agentes infecciosos, para uma resposta imune específica, além de manter uma memória imunológica com o objetivo de evitar o restabelecimento da doença, assim se aperfeiçoando a cada encontro com um invasor desconhecido (DE CASTRO, 2001).

Outro princípio utilizado nesse trabalho é o princípio da diversidade imunológica. Este princípio consiste na geração e diversificação de anticorpos dentro do repertório, distribuindo a produção de anticorpos de acordo com os agentes infecciosos conhecidos. Evitando que sejam produzidos muitos anticorpos para um tipo de agente infeccioso e poucos anticorpos para outro tipo (DE CASTRO, 2001).

2.4 REDE NEURAL ARTIFICIAL

Nessa seção são apresentados conceitos básicos de RNA, que é a técnica principal desse trabalho, a técnica que se busca melhorar com a aplicação do algoritmo SAND. São apresentados a seguir tópicos como histórico, relação entre o neurônio artificial e o neurônio biológico, e características principais como arquitetura, métodos de treinamento e funções de ativação.

2.4.1 Introdução

As Redes Neurais Artificiais surgiram por volta de 1943 por meio do trabalho de Warren McCulloch e Walter Pitts, no qual foi desenvolvido o primeiro modelo matemático (Figura 1) de um neurônio biológico (Figura 2) (DE CASTRO, 2001; SILVA, 2011). Pode-se associar o modelo matemático apresentado pela Figura 1, ao neurônio biológico apresentado pela Figura 2, da seguinte forma:

- entradas – dendritos: recebem a saída emitida por outro neurônio;
- pesos das conexões – sinapses: pesos utilizados para ponderar as entradas;
- Σ e função de ativação – soma: realiza a soma ponderada das entradas e retorna seu nível de ativação; e
- saída – axônio: transmite ou retorna seu nível de ativação, que será utilizado como entrada para outro neurônio.

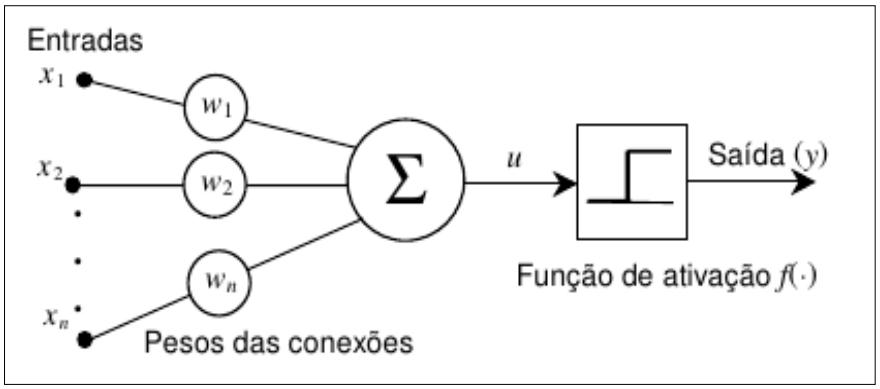

Figura 1 - Representação funcional de um Neurônio Artificial
Fonte: De Castro (2001) PÁG. 85

As RNAs são consideradas como generalizações matemáticas considerando que (DE CASTRO, 2001):

- o elemento básico é o neurônio, onde ocorre o processamento da entrada ou informação;
- a ligação entre os neurônios acontece através de conexões que transmitem a saída de um neurônio a entrada de outros;

- para cada conexão recebida é associado um peso que tem a função de ponderar a entrada recebida; e
- cada neurônio gera a saída a partir de uma função de ativação que utiliza a soma ponderada das entradas como parâmetro para retornar seu nível de ativação.

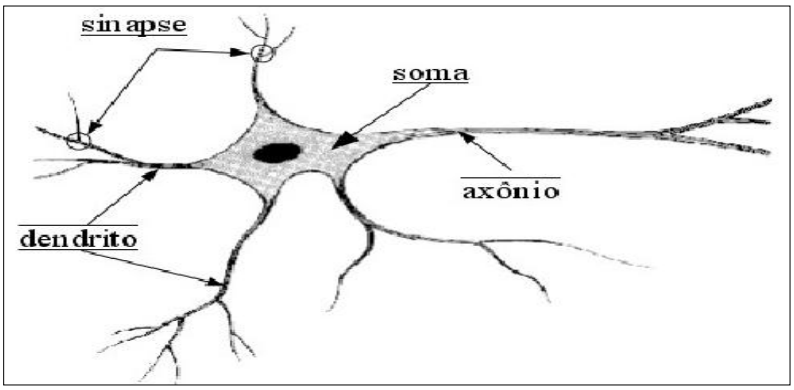

Figura 2 - Representação de um Neurônio Biológico
Fonte: Silva (2011) PÁG. 45

2.4.2 Características principais

As RNAs basicamente são caracterizadas de acordo com sua arquitetura ou estrutura, algoritmo de treinamento ou aprendizagem e função de ativação; e possuem parâmetros para inicialização da rede como número de neurônios por camada, taxa de conexão que define o número de conexões que existirá na rede, taxa de aprendizado que define o grau de agressividade do processo de treinamento, e *momentum* que pode ser usado para definir a velocidade do treinamento; e parâmetros de treinamento como margem de erro desejada, número de épocas para treinamento e número de iterações entre os relatórios gerados (FANN, 2012). Nas próximas seções as características são apresentadas de forma resumida com o objetivo de contextualizar o leitor quanto a termos utilizados mais a frente.

2.4.2.1 Arquitetura ou estrutura

Quanto a sua arquitetura ou estrutura, ou seja, a forma como são realizadas suas conexões, as RNAs pode ser caracterizadas basicamente por três classes (DE CASTRO, 2001; SILVA, 2011):

1. redes *feedforward* de única camada: caso mais simples, nas quais existem apenas a camada de entrada alimentando a última camada, ou seja, a camada de entrada simplesmente envia a entrada recebida, para a próxima camada. Essa arquitetura não permite a utilização de generalizações muito complexas, sendo adequada para trabalhar com situações de menor complexidade;
2. redes *feedforward* de múltiplas camadas: se diferem da arquitetura anterior devido a utilização de outras camadas entre a camada de entrada e a camada de saída. Essas camadas são conhecidas como camadas ocultas ou camadas intermediárias. A utilização dessas camadas aumenta o poder de generalização da rede neural, possibilitando trabalhar com situações de maior complexidade; e
3. redes recorrentes: se diferem das outras arquitetura devido a utilização de *loops*, na qual um neurônio utiliza sua própria saída, ou seja, sua própria ativação, como uma entrada.

2.4.2.2 Algoritmo de treinamento ou aprendizagem

Quanto ao algoritmo de treinamento e aprendizagem, ou seja, quanto ao processo utilizado para adaptar suas conexões ou sinapses, as RNAs pode ser caracterizadas basicamente por duas classes (DE CASTRO, 2001; SILVA, 2011):

1. supervisionada: consiste no treinamento com utilização de conjunto de dados de entrada e saída, por exemplo a base de dados NSL-KDD (2009) utilizada no trabalho de Silva (2011). Base que dados de

conexões de rede compostas por 41 campos de entrada, como tentativas de login sem sucesso; e 1 campo de saída informando se a conexão representava um ataque ou não; e

2. não-supervisionada ou auto-organizada: consiste no treinamento sem utilização de um supervisor, ou um campo no conjunto de testes referente a saída que deferia ser retornada, a rede se adapta utilizando dados estatísticos para a construção de moldes, categorias, ou representações para as entradas.

Esse processo de treinamento não se limita apenas ao ajuste de sinapses e conexões, existem trabalhos que apresentam ajustes quanto à arquitetura e funções de avaliação da RNA, como o trabalho de Silva (2011) que utiliza Algoritmos Genéticos para buscar a melhor configuração de RNA para trabalhar com bases de dados de ataque, e trabalhos citados por De Castro (2001).

2.4.2.3 Função de ativação

Como citado anteriormente na seção 2.4.1, a função de ativação é a função responsável pelo recebimento da soma ponderada das entradas e pela geração do grau de ativação do neurônio. A função de ativação pode ser definida livremente, no sentido de que cada neurônio pode utilizar uma função de ativação diferente, independente da camada a que pertence, ou pode ser definida uma função de ativação para toda uma camada, exceto a camada de entrada que tem a função de simplesmente transmitir a entrada recebida, como citado anteriormente na seção 2.4.2.1 (FANN, 2012).

2.5 ALGORITMOS GENÉTICOS

Nessa seção são apresentados conceitos básicos de Algoritmos Genéticos, que é a técnica utilizada em Silva (2011) para definição das melhores configurações de RNA para trabalhar com a base de dados NSL-KDD. Devido ao fato desse trabalho não utilizar essa técnica diretamente, são apresentados apenas conceitos básicos, como uma introdução e conceito de funcionamento da técnica.

2.5.1 Introdução

Os Algoritmos Genéticos são uma subárea da Computação Evolutiva que surgiu na década de 1950, inspirando-se em teorias de genética e evolução natural (SILVA, 2011). Sendo assim, os Algoritmos Genéticos são uma classe de algoritmos que simulam o processo de seleção natural e herança genética, realizando mutações em várias gerações com o intuito de otimizar uma solução, em termos de seleção natural, buscar o indivíduo mais forte a partir de gerações de uma população ou repertório inicial (DE CASTRO, 2001; SILVA, 2011).

2.5.2 Funcionamento

Seu funcionamento consiste na definição de uma população e a realização de uma avaliação para cada indivíduo da mesma, com o objetivo de encontrar e selecionar alguns dos indivíduos mais fortes. Uma vez que esses indivíduos foram selecionados, inicia-se o processo de reprodução que consiste na mutação ou cruzamento desses indivíduos para gerar uma nova população (SILVA, 2011).

A mutação consiste na escolha de determinado indivíduo da população e na perturbação de um ou mais genes desse mesmo indivíduo, ou seja, realizar mutação de ponto único ou mutação de múltiplos pontos. Pensando no indivíduo como um vetor de números reais, podemos considerar os genes como as posições do vetor, sendo mutação de ponto único, a mutação em apenas uma posição do vetor, enquanto a mutação de múltiplos pontos é a mutação de duas ou mais posições do vetor. O cruzamento consiste na troca de parte de determinado indivíduo com parte de outro. Considerando o exemplo anterior, sendo o indivíduo um vetor de números reais, pode-se considerar esse processo como a troca dos valores de um determinado vetor a partir da segunda posição, pelos valores de um segundo vetor a partir da segunda posição. Após esse processo de reprodução, seja mutação ou cruzamento, volta-se ao passo de avaliação até que uma condição de parada seja satisfeita (SILVA, 2011).

2.6 LÓGICA NEBULOSA

Nessa seção são apresentados conceitos básicos de Lógica Nebulosa, que é a técnica utilizada em Silva (2011) para tratar os resultados da RNA. Assim como com Algoritmos Genéticos, esse trabalho não utiliza diretamente essa técnica. Sendo assim, são apresentados apenas conceitos como histórico, objetivo e funcionamento.

2.6.1 Introdução

O conceito de Lógica Nebulosa (*Fuzzy Logic*) surgiu por volta de 1965 por meio do trabalho de Lofti A. Zadeh, no qual o mesmo observou que não era possível automatizar tarefas que necessitavam de valores de indecisão, valores

intermediários entre pertinência total e não pertinência. Tarefas encontradas em áreas como Biologia, Química, entre outras (DE CASTRO, 2001; SILVA, 2011).

2.6.2 Funcionamento

Para utilização do sistema nebuloso é necessário transformar os dados em dois processos. O primeiro processo é chamado fuzzificação que consiste na transformação de dados quantitativos para dados qualitativos através uma função de pertinência, formando um conjunto de números reais dentro do intervalo de 0 e 1. O segundo processo é chamado defuzzificação que consiste em voltar os dados qualitativos para quantitativos (SILVA, 2011).

2.7 SAND (Simulated ANnealing approach for Diversity)

Nessa seção é apresentado o algoritmo SAND, que tem o objetivo de inicializar os pesos de RNA, que é a técnica principal desse trabalho. São apresentados a seguir tópicos como histórico, fundamentos, representações, aplicações, descrição do algoritmo e seus pontos principais.

2.7.1 Introdução

O algoritmo SAND (Simulated ANnealing approach for Diversity) foi proposto pelo trabalho de De Castro e Von Zuben (1999). O algoritmo é baseado no algoritmo Simulated Annealing e aborda uma das principais características do sistema imunológico, o princípio de diversidade imunológica apresentado anteriormente na

seção 2.3.2, que consiste na geração de uma população de anticorpos com ampla cobertura do espaço de buscas. O algoritmo é adaptado para trabalhar com duas representações diferentes para anticorpos, a forma de cadeias binárias, conhecida como espaço de formas de Hamming; e a forma de vetores reais, conhecido como espaço de formas Euclidiano (DE CASTRO, 2001).

O SAND busca diversidade a partir de mutações dentro do repertório inicial, dessa forma não utiliza nenhuma informação prévia sobre o problema, da mesma forma que acontece no sistema imunológico, tornando a busca de diversidade totalmente independente do problema ou ambiente no qual o repertório será aplicado. Um exemplo de diversificação utilizando o SAND apresentado em De Castro (2001) pode ser visualizado na Figura 3 e na Figura 4. As figuras apresentam uma população composta por nove indivíduos de comprimento dois, no caso duas dimensões, representados através de espaço de formas Euclidiano, devido à utilização de números reais para representar os indivíduos. Inicialmente (Figura 3) existem indivíduos muito próximos, apresentando assim baixa cobertura do espaço de busca. Após a utilização do algoritmo (Figura 4), os indivíduos são distribuídos apresentando maior cobertura do espaço de busca, ou seja, uma população mais diversificada.

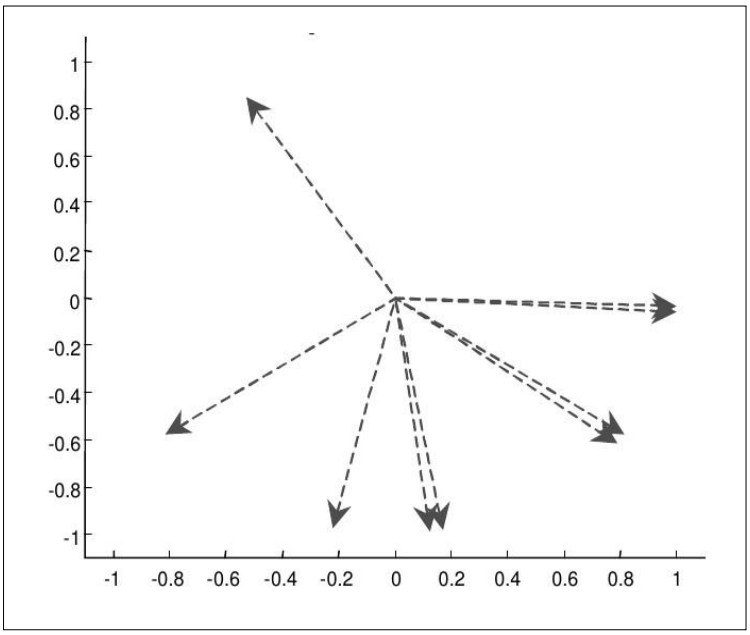

Figura 3 - Distribuição inicial de nove indivíduos

Fonte: De Castro (2001) PÁG. 138

Esse algoritmo pode ser aplicado a diversos problemas e técnicas que necessitem de otimização de condições iniciais, como inicialização de pesos de RNAs, populações de cromossomos para Algoritmos Genéticos, entre outros. Considerando RNAs de aprendizado supervisionado por exemplo, a inicialização dos pesos da rede geralmente é realizada através da geração de números randômicos respeitando determinado limite. A utilização desses pesos aleatórios pode afetar de forma negativa no processo de treinamento e no resultado apresentado pela RNA. No treinamento, uma possível baixa cobertura do espaço de buscas resulta em maior custo de treinamento. No resultado, os pesos aleatórios podem “prender” a RNA a um ótimo local (DE CASTRO, 2001).

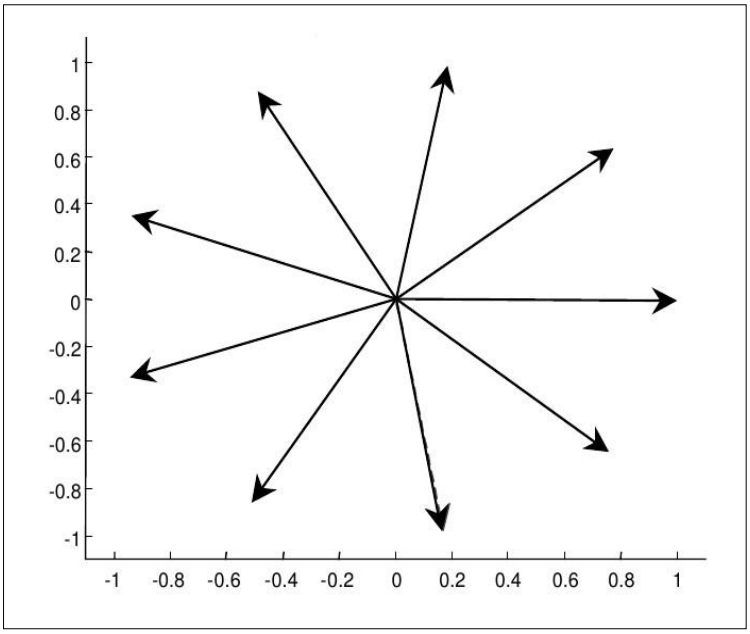

Figura 4 - Distribuição dos nove indivíduos após a aplicação do SAND
Fonte: De Castro (2001) PÁG. 138

2.7.2 Simulated Annealing

Segundo De Castro (2001), a origem desse método é associada a propriedades de alguns líquidos e sólidos, que apresentam variações de comportamento quando expostos a diferentes temperaturas. Seu processo consiste no aquecimento do sistema a uma temperatura muito elevada, o que resultará em

variações no comportamento, e logo após na redução da temperatura lentamente até que o sistema não apresente mais variações.

Para realização desse processo, o algoritmo realiza iterações de perturbações ou mutações em determinado indivíduo da população analisando o resultado encontrado. Caso o resultado da perturbação seja positivo, essa nova população é aceita e utilizada como população inicial da próxima iteração. Caso seja negativo, a utilização dessa nova população é tratada através de probabilidade, na qual a utilização dessa população é possível para qualquer temperatura não nula. Essa característica garante que o método não fique preso em ótimos locais, pois a possível utilização de populações de resultado negativo pode ocasionar mais a frente, no encontro do ótimo global (DE CASTRO, 2011).

2.7.3 Descrição do algoritmo

A seguir é apresentada uma descrição do algoritmo SAND apresentada por De Castro (2001). Vale ressaltar que a descrição abaixo é voltada para uma abordagem imunológica do algoritmo, sendo assim, seus componentes são relacionados com componentes do sistema imunológico.

Considerando a seguinte notação:

- *T*: temperatura da configuração;
- *Ab*: população de *N* anticorpos;
- *Abt*: população temporária gerada a partir de *Ab*;
- *E*: energia da população *Ab*;
- *Et*: energia da população *Abt*;
- *ct*: quantidade de iterações em estado estacionário;
- α: taxa de mutação a ser aplicada na geração de *Abt*;
- β: decrescimento geométrico da temperatura; e
- δ: limiar de iterações em estado estacionário.

A descrição do algoritmo é:

1. Gerar de uma população *Ab* inicial de *N* anticorpos;
2. Calcular a energia *E* dessa população *Ab*;
3. Enquanto determinado percentual de diversidade e determinado número de gerações não forem alcançados:
4. Provocar uma mutação no repertório *Ab*, proporcional a α, gerando uma população temporária *Abt* e reduzindo a taxa α de mutação;
5. Calcular a energia *Et* dessa população *Abt* e a variação Δ*E* entre *E* e *Et*;
    6. Caso Δ*E* < 0, aceitar a nova população *Abt* e voltar ao passo 3;
    7. Caso Δ*E* > 0, tratar probabilisticamente e voltar ao passo 3;
    8. Caso Δ*E* = 0, avaliar a quantidade de iterações *ct* em estado estacionário;
        9. Caso *ct* > *δ*, reduzir a energia a temperatura *T*, restaurar a taxa α de mutação e voltar ao passo 3;
        10. Caso *ct* <= *δ*, voltar ao passo 3, mantendo temperatura e mutação constantes;

Como podem ser observados na descrição do algoritmo, os pontos principais são a geração da população inicial, a forma como fazer as mutações, como analisar o resultado da população gerada, as condições de parada do algoritmo e a forma que será utilizada para representar os indivíduos, uma vez que a forma na qual os indivíduos são representados, afeta na forma de analisar a população gerada, como citado por De Castro (2001).

2.8 NSL-KDD e KDDCUP'99

A NSL-KDD (2009) é uma base de dados que foi gerada a partir da KDDCUP'99 (1999) devido a problemas apontados por Tavallaee (2009), como redundância no conjunto de dados, o que poderia afetar no desempenho; e na

“inclinação” dos classificadores para esses registros redundantes, e na distribuição dos dados em grupo separados, que basicamente tem o objetivo de separar dados de treino e dados de teste. Os dados da KDDCUP'99 podem ser categorizados em três grupos (SILVA, 2011):

1. características básicas: que são dados de conexão TCP/IP;
2. características de tráfego: que são dados sobre recursos da rede; e
3. recursos de conteúdo: informações referentes a ataque R2L (*Remote to Local*) e U2R (*User to Root*).

Para melhor distribuição dos dados, a NSL-KDD dividiu os dados em três grupos (NSL-KDD, 2009; SILVA, 2011):

1. KDDTrain+: grupo que contem os dados para treinamento;
2. KDDTest+: grupo que contem os dados para teste; e
3. KDDTest-21: grupo que contem dados para teste que não foram bem classificados.

Cada trecho de conexão da NSL-KDD possui 41 entradas e somente uma saída, que informa se o mesmo é um ataque ou não.

3 METODOLOGIA

Esse trabalho objetivou analisar a utilização do algoritmo SAND para inicialização de pesos de determinada RNA para reconhecimento de padrões de ataque na base de dados NSL-KDD (2009), com o intuito de melhorar os resultados apresentados em outros trabalhos da área, como o tempo de treinamento e a taxa de acerto, utilizando como base o trabalho de Silva (2011) e o trabalho de De Castro (2001).

Para alcançar o objetivo do trabalho, inicialmente foram implementados o algoritmo SAND citado anteriormente na seção 2.7, e a rede neural apresentada em Silva (2011). Após as implementações foram realizados testes de treinamento e execução da rede neural original e das redes neurais alteradas pelo algoritmo SAND, comparando os resultados obtidos quanto à tempo de treinamento e taxa de acerto.

Nas próximas seções será detalhado o processo utilizado para realização do trabalho. O processo é dividido em três seções:

- implementações: descreve linguagem e biblioteca utilizadas, justificando sua escolha e descreve a implementação do algoritmo SAND e da RNA;
- testes: descreve os testes realizados, apresentando casos, importâncias e justificativas; e
- dados: descreve os dados utilizados no teste, apresentando tipo de dado, quem coletou e como coletou, e justifica sua utilização.

3.1 IMPLEMENTAÇÕES

Inicialmente foi realizado um estudo sobre linguagens e bibliotecas disponíveis para desenvolvimento do trabalho. Esse estudo se baseou em outros trabalhos da área como o trabalho de Silva (2011) e Uchôa (2009) com objetivo de

conhecer as linguagens e ferramentas utilizadas pelos autores. Uma vez que a linguagem e biblioteca foram apontadas pelos autores, foi realizado um estudo das documentações da linguagem e biblioteca, com o objetivo de conhecer vantagens e desvantagens da utilização das mesmas.

A linguagem escolhida para realização desse trabalho foi a linguagem Python (2012). O que justifica a escolha dessa linguagem é o fato de ser utilizada em outros trabalhos da área, ser uma linguagem livre, possuir características que facilitam os testes e verificações das implementações, como ser interpretada (PYTHON, 2012); e devido a existência de bibliotecas para trabalhar com RNA, como a biblioteca FANN (*Fast Artificial Neural Network Library*).

A biblioteca escolhida foi a biblioteca FANN (2012), que é uma biblioteca livre para desenvolvimento de RNAs, implementada em linguagem C com suporte a mais de 15 linguagens de programação, como Python, Ruby, Prolog, entre outras, e possui documentação muito completa, descrevendo além de suas funções, conceitos de RNA (FANN, 2012).

Após a definição de linguagem (PYTHON, 2012) e biblioteca (FANN, 2012), foram implementados o algoritmo SAND e a RNA, que serão detalhados nas próximas seções.

3.1.1 Algoritmo SAND

O algoritmo foi desenvolvido seguindo a descrição apresentada por De Castro (2001). Dessa forma, nessa seção serão apresentadas as configurações utilizadas para os pontos principais citados anteriormente na seção 2.7.3. Sendo pontos principais:

- a forma de representar e gerar a população inicial;
- a forma de realizar as mutações;
- como analisar os resultados da mutação; e
- as condições de parada.

Para mais detalhes quanto a implementação do algoritmo, pode ser encontrado no “ANEXO I – Biblioteca SAND” o código fonte da biblioteca, onde foram implementadas todas as funções utilizadas no processo de diversificação do algoritmo SAND.

3.1.1.1 Representação dos indivíduos e população inicial

Como citado anteriormente, o algoritmo foi aplicado a uma RNA para inicialização de seus pesos. Dessa forma a população inicial é o conjunto de pesos gerados no momento de criação da RNA com a biblioteca FANN (2012). Por padrão, a biblioteca FANN (2012) inicializa os pesos da RNA com valores reais gerados de forma aleatória dentro do intervalo de -0,1 e 0,1. Sendo assim não foi necessário gerar uma população inicial, essa tarefa foi realizada pela biblioteca FANN (2012) na geração da RNA. A representação dos indivíduos é definida de acordo com a população inicial, e como a população é um conjunto de pesos de RNA, os indivíduos são representados por vetores de valores reais, ou seja, espaço de formas Euclidiano.

Para mais detalhes sobre a utilização e obtenção das populações e dos indivíduos, pode ser encontrado no “ANEXO II – Criação da Rede Neural Artificial” e no “ANEXO III – Aplicação do SAND sobre a Rede Neural Artificial” o código utilizado para geração, e aplicação do SAND sobre a RNA com a biblioteca FANN (2012).

3.1.1.2 Mutações

Para realização das mutações foram utilizados os conceitos de mutações de ponto único e múltiplos pontos, apresentados anteriormente na seção 2.5.2. Para escolha do indivíduo para mutação, o algoritmo buscava dentro da população o par

de indivíduos mais próximos. Para encontrar esse par de indivíduos era necessário pesquisar todo o repertório e calcular a distância euclidiana[2] entre todos esses indivíduos. Dentre esse par de indivíduos, o algoritmo selecionava um indivíduo e um ou mais genes de forma aleatória para sofrer mutação. Uma vez que o indivíduo e os genes foram selecionados, o algoritmo gerava um valor aleatório dentro do intervalo de -1 a 1 para somar ao valor desses genes.

O que justifica esse processo de seleção de indivíduos para mutação, é a necessidade de aumentar a diversidade da população. Outra forma de selecionar um indivíduo para mutação seria simplesmente escolher de forma aleatória um indivíduo e aplicar a mutação. Ambas as opções são válidas, e apresentam vantagens e desvantagens. A seleção utilizada, de encontrar o par de indivíduos mais próximos e aplicar a mutação sobre eles é mais vantajosa no sentido de que dessa forma, nunca efetuaríamos mutações em indivíduos que estivessem mais distantes, ou seja, mais diversificados, otimizando assim a diversificação da população e evitando um afastamento excessivo de determinado indivíduo. Porém existe a desvantagem de que a cada mutação é necessário buscar todas as distâncias entre os indivíduos da população antes de selecionar um indivíduo, gerando assim maior custo para a mutação. A seleção aleatória é mais vantajosa no sentido de não precisar buscar todas as distâncias já que o indivíduo seria selecionado de forma aleatória, gerando assim menor custo para mutação. Porém existe a desvantagem no sentido que a seleção de indivíduo de forma aleatória poderia resultar na escolha de um indivíduo que já estivesse distante ou diversificado, resultando no afastamento excessivo desse indivíduo.

Esse processo de mutação, a forma de escolher um indivíduo, o número de pontos para mutação, gera certa indecisão sobre o que seria mais adequado ou eficaz. Para realização desse trabalho, o processo escolhido é o processo que busca otimizar a diversificação evitando possível afastamento excessivo, ao custo de a cada mutação verificar todas as distâncias dentro do repertório.

Para mais detalhes de como foram realizadas as mutações e como foram calculadas as distâncias entre os indivíduos, pode ser encontrado no “ANEXO I – Biblioteca SAND” o código utilizado para definir a biblioteca, onde são

[2] distância euclidiana: distância entre dois pontos ou anticorpos seguindo a abordagem imunológica, calculada através da equação: $\sqrt{\sum_{i=1}^{L}(Ab_i - Ag_i)^2}$. Na qual *L* é o tamanho dos elementos, *i* é a posição corrente e *Ab* e *Ag* são os elementos.

implementadas as funções para realização da mutação e cálculo da distância euclidiana.

3.1.1.3 Análise dos resultados da mutação

Como citado anteriormente na seção 2.7.3, a forma de analisar os resultados da mutação depende da forma em que os indivíduos da população foram representados. Para analisar os resultados de indivíduos utilizando espaço de formas Euclidiano, que é a representação utilizada nesse trabalho, é necessário calcular a amplitude do vetor médio, ou seja, o percentual de diversificação da população. Esse percentual de diversificação pode ser obtido através da distância euclidiana entre o vetor médio da população e o centro de coordenadas. Para obter o vetor médio da população, basta realizar o somatório dos indivíduos da população e dividi-lo pelo total de indivíduos. Uma vez que o vetor médio foi calculado, basta calcular a distância euclidiana entre esse vetor e o centro de coordenadas, distância que é utilizada para calcular o percentual de diversidade (DE CASTRO, 2001).

Para mais detalhes de como foram calculados o vetor médio, a distância do vetor médio ao centro de coordenadas, e o percentual de diversidade, pode ser encontrado no “ANEXO I – Biblioteca SAND” o código utilizado para definir a biblioteca, onde são implementadas as funções para realização desses cálculos de vetor médio, distância euclidiana e percentual de diversidade.

3.1.1.4 Condições de parada

Como citado anteriormente na seção 2.7.3, as condições de parada são pontos principais, uma vez que definem dois parâmetros: número máximo de iterações ou gerações; e percentual máximo de diversidade. Para desenvolvimento desse trabalho, as duas condições são utilizadas, sendo 5000 o número máximo de

gerações e 99,5% o percentual máximo de diversidade. Ambos os parâmetros foram definidos de acordo com valores recomendados por De Castro (2011).

3.1.2 Rede Neural Artificial

Para implementação da RNA, foram utilizadas configurações apresentadas como mais adequadas para trabalhar com a base de dados NSL-KDD no trabalho de Silva (2011). Para esclarecer a RNA utilizada, a seguir serão apresentadas as configurações utilizadas e as características dessa RNA de acordo com as características apresentadas no referencial teórico.

Sendo as configurações:

- número de neurônios na camada de entrada: 41 neurônios devido ao número de entradas da base de dados NSL-KDD que será detalhada mais a frente;
- número de neurônios na camada oculta: 25 neurônios de acordo com os resultados apontados por Silva (2011);
- número de neurônios na camada de saída: 2 neurônios, probabilidade de ser um ataque e probabilidade de ser um tráfego normal;
- taxa de conexão: 1, definindo que a rede será completamente conectada;
- taxa de aprendizado: 0,3 de acordo com os resultados apontados por Silva (2011), definindo uma taxa de aprendizado pouco agressiva, uma vez que a taxa padrão da biblioteca FANN (2012) é 0,7;
- *momentum*: 0,3 de acordo com os resultados apontados por Silva (2011);
- margem de erro desejado: 0,000001;
- número de épocas para treinamento: 200 épocas de acordo com os resultados apontados por Silva (2011);
- número de iterações entre os relatórios de treinamento: 10;

- função de ativação da camada oculta: *SIGMOID_STEPWISE*, sigmóide passo a passo, de acordo com os resultados apontados por Silva (2011);
- função de ativação da camada de saída: *SIGMOID*, sigmóide, de acordo com os resultados apontados por Silva (2011); e
- algoritmo de treinamento: *Backpropagation*, de acordo com os resultados apontados por Silva (2011).

A partir das configurações apresentadas pode-se definir a RNA utilizada como uma rede *feedforward* de múltiplas camadas devido a sua disposição de 41 neurônios na camada de entrada, a existência de uma camada intermediária ou oculta com 25 neurônios e 2 neurônios na camada de saída. É uma rede que utiliza aprendizado supervisionado através da função de *Backpropagation* devido a utilização de dados para verificação das saídas e ajustes de sinapses. É uma rede que utiliza duas funções diferentes para ativação de seus neurônios, aplicando a função *SIGMOID_STEPWISE* para a camada oculta e a função *SIGMOID* para a camada de saída.

Todas as configurações apresentadas, como parâmetros de criação e treinamento, funções de ativação e treinamento foram definidas com a biblioteca FANN (2012). Para mais detalhes da implementação pode ser encontrado no “ANEXO II – Criação da Rede Neural Artificial”, no “ANEXO IV – Treinamento da Rede Neural Artificial” e no “ANEXO V – Execução da Rede Neural Artificial” o código utilizado para geração, treinamento e execução da RNA com a biblioteca FANN (2012).

3.2 TESTES

Uma vez que o algoritmo SAND e a RNA foram implementados, o próximo passo foi a realização de testes com o algoritmo SAND isoladamente, ou seja, antes de ser aplicado sobre a RNA, o SAND passou por alguns testes com o objetivo de verificar se o mesmo alcançava a diversificação esperada.

Para realização desses testes iniciais, foram utilizadas populações geradas de forma aleatória com o apoio da biblioteca SAND disponível no “ANEXO I – Biblioteca SAND”, e foi utilizada também a população apresentada anteriormente na Figura 3, que ilustra uma população composta de nove indivíduos de baixa diversidade, falando em números, uma população com 49% de diversidade. Em ambos os testes o algoritmo respondeu de forma satisfatória, alcançando um grande aumento na diversificação da população. No exemplo apresentado pela Figura 3, o algoritmo consegue aumentar o percentual de diversidade de 49% para 99% como pode ser visualizado na Figura 4, ambas apresentadas na seção 2.7.1.

Uma vez que o algoritmo conseguiu aumentar a diversidade dessas populações de forma satisfatória, o mesmo foi utilizado sobre a RNA. Para utilização do algoritmo sobre RNA, é necessário aplicá-lo para cada camada da rede neural, por exemplo, a RNA utilizada era composta por 41 neurônios na camada de entrada, 25 neurônios na camada oculta e 2 neurônios na camada de saída. Seguindo essa configuração, existem duas populações de pesos na RNA, a população da camada oculta, composta por 25 indivíduos de 41 genes, sendo 25 o número de neurônios da camada e 41 o número de neurônios da camada anterior; e a população da camada de saída, composta por 2 indivíduos de 25 genes, sendo 2 o número de neurônios da camada e 25 o número de neurônios da camada anterior. Dessa forma o algoritmo é aplicado sobre cada camada isoladamente, considerando os neurônios como indivíduos e suas entradas ou sinapses como genes.

Os testes do algoritmo sobre RNA foram realizados utilizando as duas formas de mutação citadas anteriormente na seção 2.5.2, mutações de ponto único e múltiplos pontos. Para cada tipo de mutação foram geradas 10 redes diversificadas a partir da rede original. O que justifica a utilização das duas formas de mutação é importância de realizar testes com várias formas de mutação em busca de melhores resultados, e o que justifica a geração desse número de redes para cada tipo de mutação é o fato da utilização de vários números randômicos como a escolha do indivíduo, a escolha dos genes, e taxas de mutação.

Para realização dos testes com mutação de ponto único, o algoritmo selecionava apenas uma posição do indivíduo de forma aleatória para aplicar a mutação. Para realização dos testes com mutação em múltiplos pontos, o algoritmo selecionava até 33% da população para aplicar as mutações. Dessa forma, considerando a população correspondente a camada oculta, a cada mutação até 14

genes poderiam sofrer mutação; considerando a população correspondente a camada de saída, a cada mutação até 9 genes poderiam sofrer mutação. A definição do número de genes que sofreriam a mutação era realizada para cada mutação através da geração de um número aleatório entre o intervalo de 2 e o máximo de genes de acordo com a camada, ou seja, o número de genes para mutação na população da camada oculta é definido de forma aleatória entre 2 e 14. Sendo assim na primeira mutação poderiam ser definidos 5 pontos, na segunda mutação poderiam ser definidos 2 pontos, depois 10 pontos e assim sucessivamente.

Como citado na seção 3.1.1.2, a forma como as mutações são realizadas é um ponto principal para sucesso ou fracasso da implementação. Na mutação de múltiplos pontos, a escolha de um número muito alto de genes poderia resultar no afastamento excessivo do indivíduo, dessa forma o percentual máximo de mutação nos indivíduos foi definido como 33%, valor escolhido de forma experimental, não sabendo ao certo se os resultados seriam positivos ou negativos.

Uma vez que essas redes foram geradas, foram realizados testes de treinamento e execução para cada rede, com o objetivo de obter os resultados de treinamento e execução para posterior comparação com a rede original. Os resultados apresentados pelos testes de treinamento e execução, e a análise desses resultados serão apresentados no próximo capítulo.

3.3 DADOS

Para realização dos testes de treinamento e execução das redes, foi utilizada a base de dados NSL-KDD (2009) do MIT (*Massachusetts Institute of Technology*) apresentada na seção 2.8.

O que justifica a utilização dessa base é o fato de ter sido criada a partir de dados reais de conexões de rede e ser a base utilizada no trabalho de Silva (2011), que é uma das referências principais desse trabalho. Mantendo assim a fidelidade ao ambiente utilizado inicialmente, uma vez que com a utilização de outra base de dados, existiria outro ambiente, o que impossibilitaria a comparação entre os

resultados obtidos. A obtenção da base de dados pode ser realizada em NSL-KDD (2009).

4 RESULTADOS

Como citado anteriormente na seção 3.2, foram geradas 10 redes neurais a partir da configuração inicial, para cada tipo de mutação. Vale ressaltar que essas 10 redes não sofreram nenhuma alteração em sua estrutura, funções de treinamento e avaliação, a única alteração realizada foi o ajuste dos pesos pelo algoritmo SAND. Os resultados obtidos para cada tipo de mutação são apresentados isoladamente nas próximas seções. Para cada tipo de mutação são apresentados os resultados de diversificação, de treinamento e de execução. Ao final é apresentada uma seção comparando os resultados obtidos para mutação em ponto único e mutação em múltiplos pontos.

4.1 MUTAÇÃO DE PONTO ÚNICO

Nessa seção são apresentados os resultados de diversificação, treinamento e execução para os testes com mutação de ponto único, apresentada anteriormente na seção 2.5.2, que consiste na seleção de apenas um gene do indivíduo de forma aleatória para aplicação da mutação. Ao final é apresentado um resumo dos resultados obtidos com mutação de ponto único.

4.1.1 Diversificação

Os resultados de diversificação foram obtidos através da aplicação do algoritmo SAND com mutações de ponto único sobre a RNA original para geração de 10 novas redes diversificadas. Dentre as 10 novas redes, o algoritmo alcançou aumento de diversidade entre 9,03 e 19,18 pontos percentuais na camada oculta, e

alcançou melhorias de diversidade entre 73,99 e 75,19 pontos percentuais na camada de saída. Vale lembrar que a camada oculta era composta por 25 neurônios e a camada de saída era composta por 2 neurônios. Sendo assim, o que justifica a diferença de diversidade da RNA original para as RNAs diversificadas é o número de neurônios de cada camada, ou seja, o número de indivíduos de cada população. A população da camada oculta era composta de 25 neurônios, enquanto a população da camada de saída era composta por apenas 2 neurônios, logo a diversidade inicial dos indivíduos da camada oculta é consideravelmente melhor que a diversidade da camada de saída. Sendo assim, enquanto a diversidade na camada oculta apresentou um aumento de 78,78% para 97,96%, a diversidade na camada de saída apresentou um aumento de 24,35% para 99,54%, ou seja, quanto menor a população, mais eficaz é o processo de diversificação.

Para facilitar o entendimento, a Tabela 1 e o Gráfico 1 apresentam os resultados obtidos durante o processo de diversificação. Ambos apresentam a rede original (RNA Origem) e as 10 novas redes (RNA número da rede) com seus respectivos percentuais de diversidade por camada.

Tabela 1 - Diversidade das redes por camada com mutação em ponto único

RNA	**Camada oculta**	**Camada de saída**
Origem	78,78%	24,35%
1	88,64%	99,44%
2	88,61%	99,54%
3	89,59%	99,51%
4	87,81%	99,20%
5	88,88%	99,14%
6	97,96%	99,32%
7	89,69%	98,34%
8	90,39%	99,30%
9	89,63%	99,43%
10	90,55%	99,44%

Como pode ser visualizado na Tabela 1, a melhor diversidade alcançada foi a diversidade da RNA 6, que apresentou um aumento de 19,18 pontos percentuais na

camada oculta e 74,97 pontos percentuais na camada de saída. Vale lembrar que os parâmetros utilizados como condições de parada são apresentados na seção 3.1.1.4, sendo 5000 o número máximo de gerações e 99,5% o percentual máximo de diversidade.

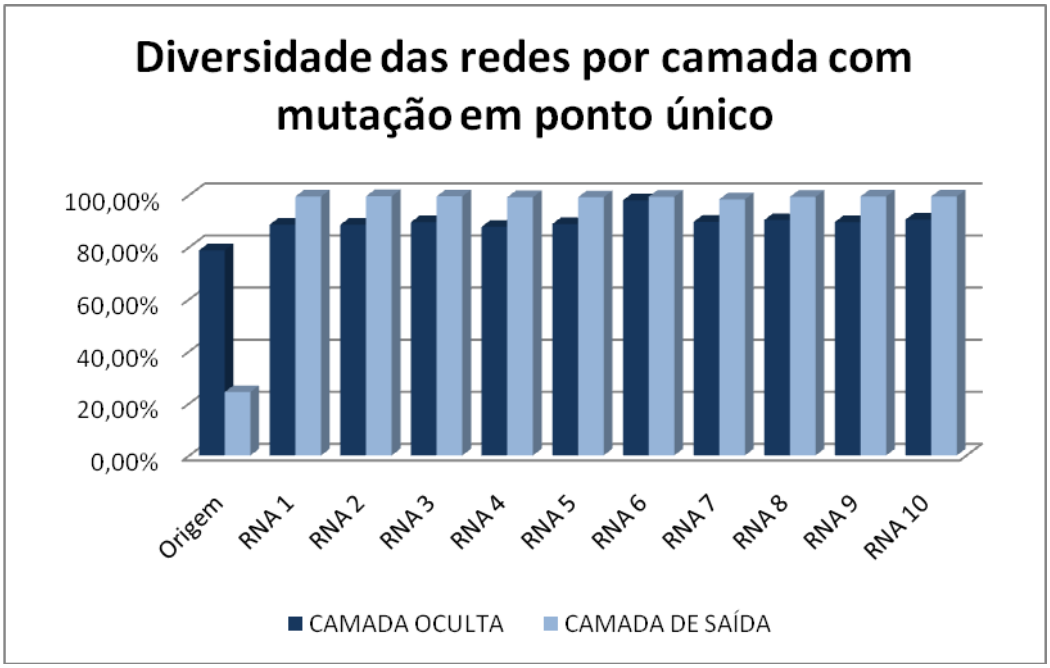

Gráfico 1 - Diversidade das redes por camada com mutação em ponto único

Considerando esses parâmetros e a camada oculta, se pode concluir que o algoritmo não alcançou o máximo de diversidade em nenhuma rede, ou seja, o critério de parada utilizado foi o máximo de gerações (Tabela 1). Considerando a camada saída, se pode concluir que o algoritmo conseguiu alcançar o máximo de diversidade em apenas duas redes RNA 2 e RNA 3 (Tabela 1). Apesar de alcançar a diversidade máxima em poucos casos, se pode notar que o algoritmo alcança aumento significativo quanto à diversidade das populações como pode ser visualizado através da Tabela 1 e do Gráfico 1.

4.1.2 Treinamento

Os resultados de treinamento foram obtidos por meio da aplicação das RNAs sobre a base KDDTrain+_20Percent, que representa 20% da base KDDTrain+ que contem dados de treinamento, apresentada na seção 2.8.

Após o treinamento das redes sobre a base a KDDTrain+_20Percent, as mesmas eram testadas sobre a base KDDTest+ com o intuito de obter o MSE (*Mean Square Error*), ou seja, o erro médio quadrado da rede (FANN, 2012).

Dentre as 10 novas redes, o algoritmo em nenhum caso conseguiu reduzir o número de épocas de treinamento e em apenas dois casos conseguiu reduzir o MSE da rede original.

Para facilitar o entendimento, a Tabela 2 e o Gráfico 2 apresentam os resultados obtidos durante o processo de treinamento. Ambos apresentam a rede original (RNA Origem) e as 10 novas redes (RNA número da rede) com seus respectivos valores de MSE e números de épocas de treinamento.

Tabela 2 - Treinamento das redes de ponto único sobre base KDDTrain+_20Percent

RNA	Épocas	MSE
Origem	200	172×10^{-3}
1	200	175×10^{-3}
2	200	168×10^{-3}
3	200	188×10^{-3}
4	200	190×10^{-3}
5	200	188×10^{-3}
6	200	171×10^{-3}
7	200	196×10^{-3}
8	200	220×10^{-3}
9	200	198×10^{-3}
10	200	176×10^{-3}

Como pode ser visualizado na Tabela 2, o melhor valor de MSE alcançado foi o valor da RNA 2, que reduziu em 0,004 o MSE da RNA original; seguido pela RNA 6,

que reduziu em 0,001 o MSE da RNA original. Vale lembrar que os parâmetros utilizados para treinamento são apresentados na seção 3.1.2, sendo 0,000001 a margem de erro desejada, 200 o número máximo de épocas de treinamento, e 10 o número de iterações entre os relatórios de treinamento.

Considerando esses parâmetros, se pode concluir que o algoritmo não conseguiu reduzir o número de épocas de treinamento em nenhuma rede. Quanto a MSE, o algoritmo conseguiu reduzir a margem em apenas duas redes RNA 2 e RNA 6 (Tabela 2 e Gráfico 2). A partir dos dados apontados pela Tabela 2 e pelo Gráfico 2, se pode notar que o algoritmo não gerou grandes resultados para o processo de treinamento, apresentando pequenas melhorias de MSE em apenas dois casos.

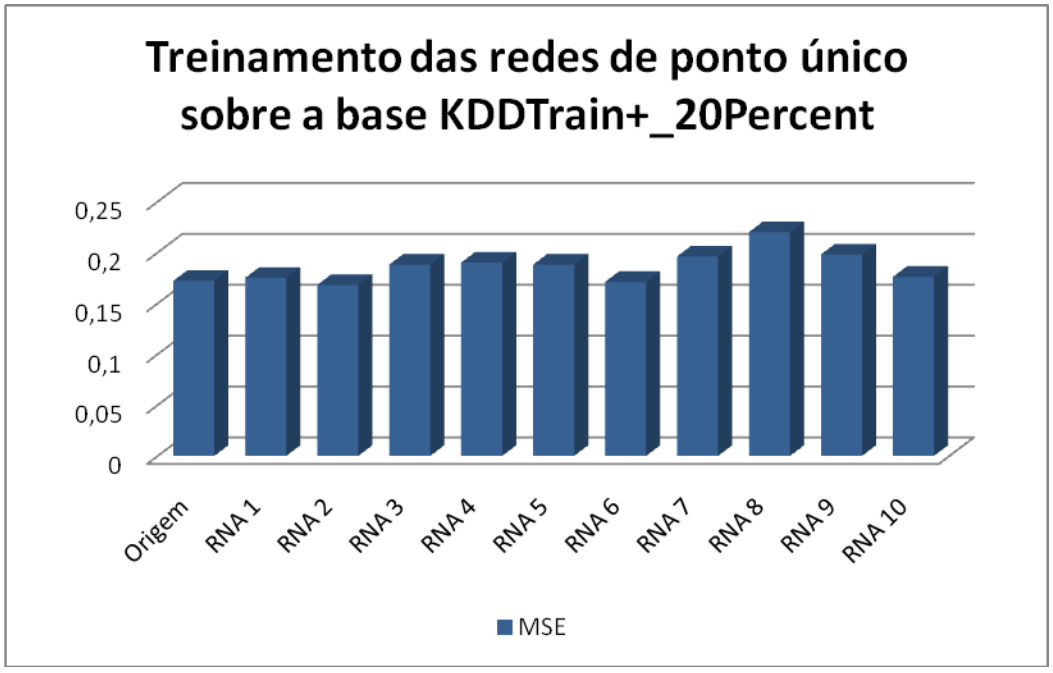

Gráfico 2 - Treinamento das redes de ponto único sobre a base KDDTrain+_20Percent

4.1.3 Execução

Os resultados de execução foram obtidos através da aplicação das RNAs sobre as 3 bases da NSL-KDD apresentadas na seção 2.8. Sendo elas:

- KDDTrain+: base que contem os dados de treinamento;

- KDDTrain+_20Percent: base que contem 20% dos dados da base KDDTrain+;
- KDDTest+: base que contem os dados para teste; e
- KDDTest-21: base que contem os dados para teste que não foram bem classificados.

Dentre as 10 novas redes, 4 redes apresentaram resultados positivos quanto a taxa de acerto. Apresentando melhorias entre 1% e 3% na base KDDTest-21, e melhorias entre 1% e 2% na base KDDTest+. As demais redes apresentaram resultados negativos quanto à taxa de acerto. Apresentando piora entre 1% e 8% na base KDDTest-21, e piora entre 1% e 4% na base KDDTest+. Vale lembrar que a base KDDTrain+_20Percent representa 20% da base KDDTrain+, e foi utilizada durante o processo de treinamento, o que justifica a diferença significativa em taxa de acerto, alcançado resultados entre 97% e 100% de taxa de acerto.

Para facilitar o entendimento, a Tabela 3 e o Gráfico 3 apresentam os resultados obtidos durante o processo de execução. Ambos apresentam a rede original (RNA Origem) e as 10 novas redes (RNA número da rede) com suas respectivas taxas de acerto por base de dados.

Tabela 3 - Execução das redes de ponto único sobre as bases de dados

RNA	KDDTest-21	KDDTest+	KDDTrain+_20Percent	KDDTrain+
Origem	63,00%	80,00%	99,00%	99,00%
1	64,00%	81,00%	99,00%	99,00%
2	65,00%	82,00%	99,00%	99,00%
3	59,00%	78,00%	99,00%	99,00%
4	59,00%	78,00%	100,00%	99,00%
5	62,00%	80,00%	99,00%	99,00%
6	66,00%	82,00%	99,00%	99,00%
7	60,00%	79,00%	99,00%	99,00%
8	55,00%	76,00%	97,00%	97,00%
9	56,00%	77,00%	99,00%	99,00%
10	64,00%	81,00%	99,00%	99,00%

Como pode ser visualizado na Tabela 3, a melhor rede encontrada foi a RNA 6 que apresentou 3 pontos percentuais de melhoria na taxa de acerto na base KDDTest-21 e 2 pontos percentuais de melhoria na taxa de acerto na base KDDTest+; seguida pela RNA 2, que apresentou 2 pontos percentuais de melhoria na taxa de acerto na base KDDTest-21 e 2 pontos percentuais de melhoria na taxa de acerto na base KDDTest+. A pior rede encontrada foi a RNA 8 que apresentou 8 pontos percentuais de piora na taxa de acerto na base KDDTest-21 e 4 pontos percentuais de piora na taxa de acerto na base KDDTest+; seguida pela RNA 9 que apresentou 7 pontos percentuais de piora na taxa de acerto na base KDDTest-21 e 3 pontos percentuais de piora na taxa de acerto na base KDDTest+.

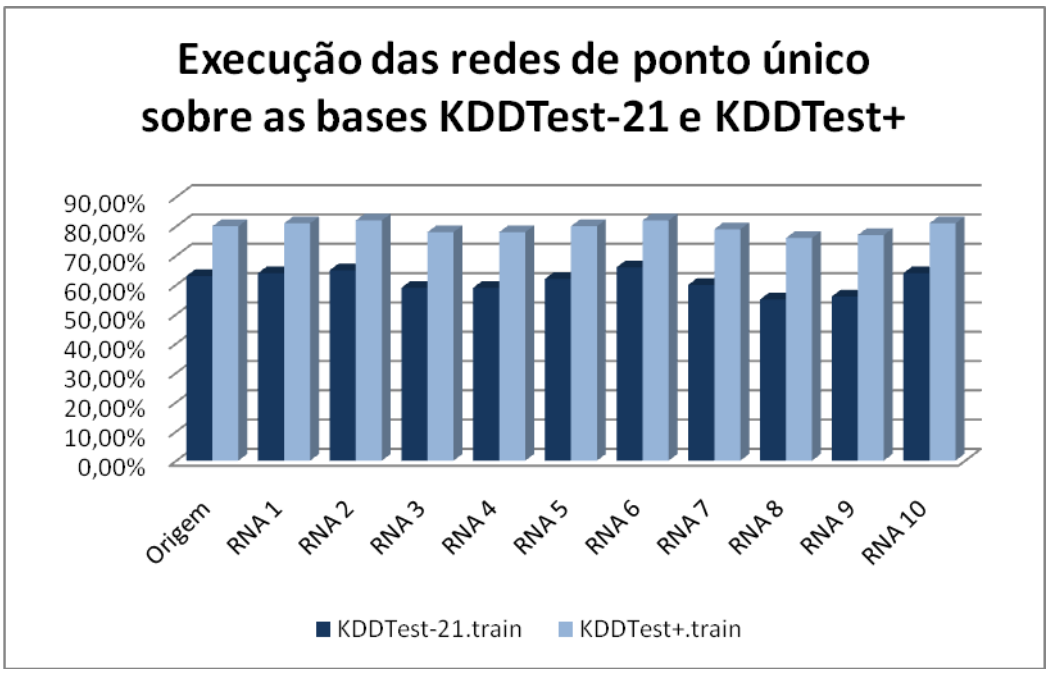

Gráfico 3 - Execução das redes de ponto único sobre as bases KDDTest-21 e KDDTest+

Considerando as taxas de acerto apresentadas pela Tabela 3 e pelo Gráfico 3, se pode concluir que o algoritmo consegue melhorar a taxa de acerto em todas as bases, mas não garante essa melhoria, uma vez que apresentou 3 e 2 pontos percentuais de melhoria nas bases KDDTest-21 e KDDTest+ com a RNA 6; e apresentou 8 e 4 pontos percentuais de piora nas bases KDDTest-21 e KDDTest+ com a RNA 8.

4.1.4 Análise geral

O processo de diversificação alcançou melhorias significativas quanto à diversidade das populações, apresentando aumento de diversidade entre 9,03 e 19,18 pontos percentuais na camada oculta e aumento de diversidade entre 73,99 e 75,19 pontos percentuais na camada saída. Porém qualquer conclusão quanto à importância de utilização do algoritmo SAND considerando apenas esse processo é precipitada, uma vez que ainda não são conhecidos os resultados de treinamento e execução. Dessa forma, o que se pode concluir até esse ponto é que o algoritmo de fato melhora a diversidade.

O processo de treinamento não alcançou melhorias tão significativas, uma vez que em nenhum caso apresentou redução das épocas e em apenas dois casos apresentou melhorias entre 0,001 e 0,004 quanto a MSE. Porém qualquer conclusão quanto à importância de utilização do algoritmo SAND até esse processo ainda é precipitada, uma vez que ainda não são conhecidos os resultados de execução. Dessa forma, o que se pode concluir até esse ponto é que o algoritmo de fato melhora a diversidade, e pode melhorar o processo de treinamento, ou seja, aumento de diversificação não implica necessariamente em melhoria no processo de treinamento.

O processo de execução foi o último sobre as bases de dados. Nesse processo foram encontrados resultados relativamente baixos, nos quais o melhor resultado apresentado foi melhoria de 3 pontos percentuais de taxa de acerto na base KDDTest-21 e melhoria de 2 pontos percentuais de taxa de acerto na base KDDTest+. Nesse ponto, se pode concluir que o algoritmo de fato melhora a diversidade, pode melhorar o processo de treinamento e pode melhorar taxa de acerto, mas não garante melhorias, ou seja, aumento de diversificação não implica necessariamente em melhoria no processo de treinamento e melhoria nas taxas de acerto. A comparação dos resultados de taxas de acerto (Tabela 3) com os resultados de diversificação (Tabela 1) gera certo impasse, uma vez que as duas melhores redes encontradas RNA 6 e RNA 2 apresentam uma diferença considerável de diversidade e apresentam taxas de acerto muito próximas.

Enquanto a RNA 6 apresenta 97,96% de diversidade (Tabela 1) e 66% de taxa de acerto na base KDDTest-21 (Tabela 3), a RNA 2 apresenta 88,61% de diversidade (Tabela 1) e 65% de taxa de acerto na mesma base. A princípio essa diferença de diversidade pode colocar em dúvida a importância do processo de diversificação, mas analisando os indivíduos da população da camada oculta da RNA 6 (Figura 5) e da RNA 2 (Figura 6) é possível visualizar semelhanças quanto a disposição dos indivíduos.

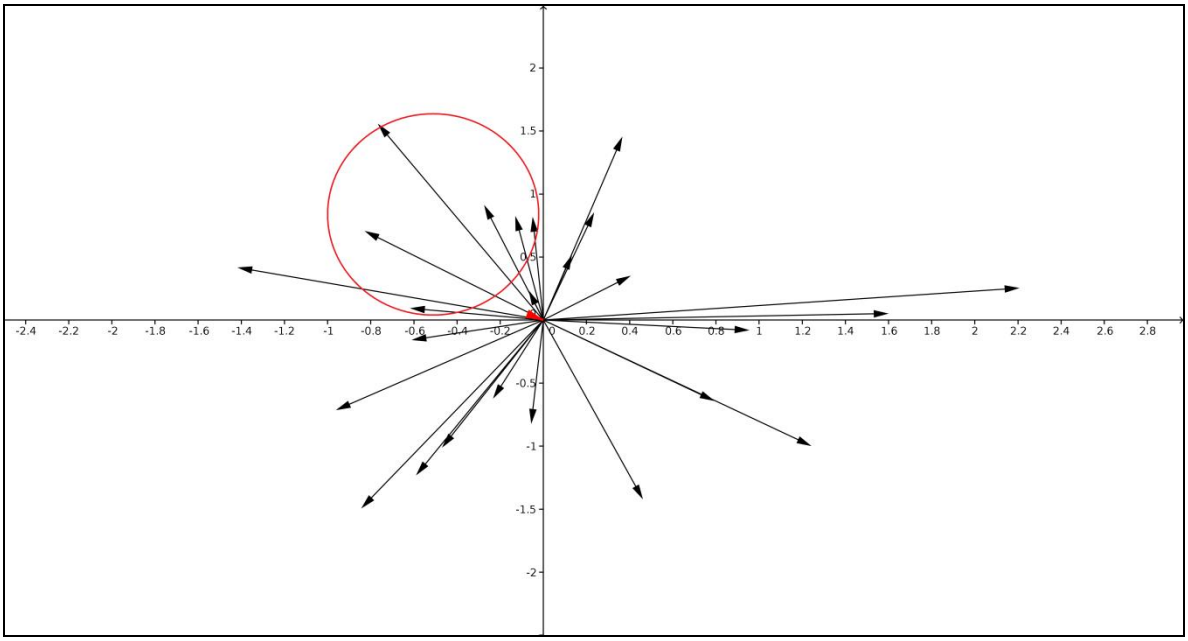

Figura 5 - População da camada oculta da RNA 6 utilizando mutação de ponto único

Em ambas as figuras (Figura 5 e 6) é possível notar uma concentração considerável de indivíduos no primeiro quadrante do plano cartesiano, no qual estão concentrados 8 indivíduos da RNA 6, e 9 indivíduos da RNA 2. Dessa forma é possível encontrar um padrão entre as duas redes, existe um número considerável de indivíduos em determinada área, apontando assim uma região promissora no espaço de buscas, uma vez que as duas redes apresentaram os melhores resultados de MSE (Tabela 2) e taxas de acerto (Tabela 3); e o mesmo não acontece para as duas redes apontadas como piores, a RNA 8 (Figura 7) e a RNA 9 (Figura 8).

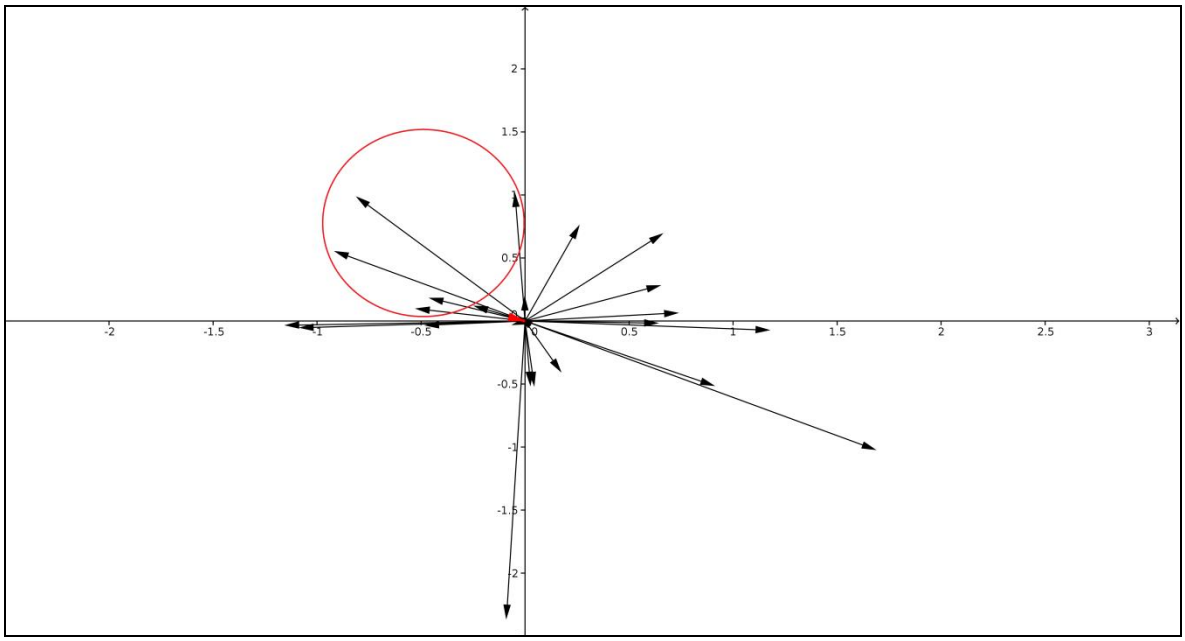

Figura 6 - População da camada oculta da RNA 2 utilizando mutação de ponto único

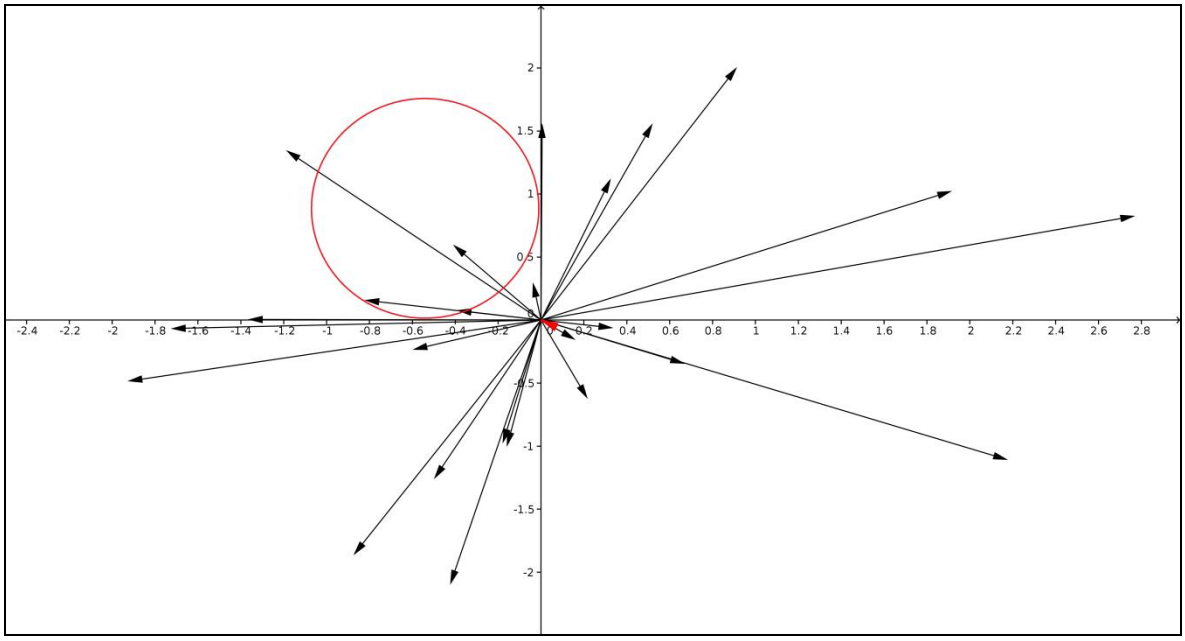

Figura 7 - População da camada oculta da RNA 8 utilizando mutação de ponto único

A utilização do algoritmo com mutações em ponto único é interessante. As melhorias no processo de treinamento não foram muito satisfatórias, porém qualquer melhoria nas taxas de acerto do processo de execução é bem vinda por menor que seja devido às deficiências apontadas nos SDIs e a necessidade constante de melhoria citadas anteriormente, e melhorias de 3 pontos percentuais de taxa de

acerto na base KDDTest-21 e 2 pontos percentuais de taxa de acerto na base KDDTest+ foram alcançadas.

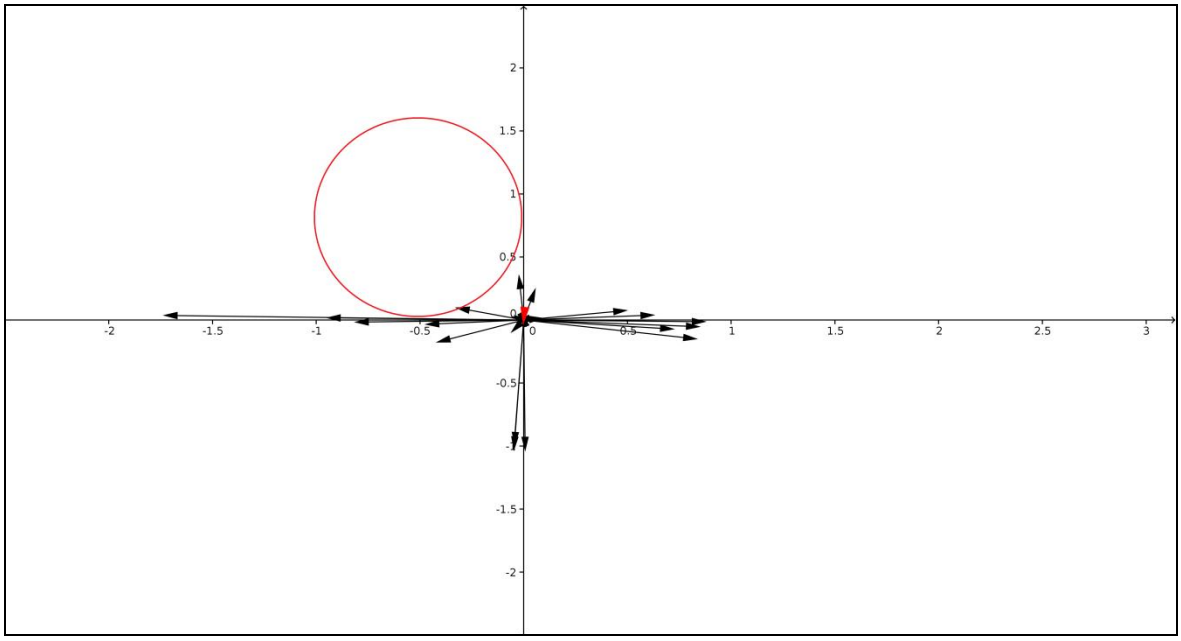

Figura 8 - População da camada oculta da RNA 9 utilizando mutação de ponto único

4.2 MUTAÇÃO EM MÚLTIPLOS PONTOS

Nessa seção são apresentados os resultados de diversificação, treinamento e execução para os testes com mutação em múltiplos pontos, apresentada anteriormente na seção 2.5.2, que consiste na seleção de dois ou mais genes do indivíduo de forma aleatória para aplicação da mutação. Ao final é apresentado um resumo dos resultados obtidos com mutação em múltiplos pontos.

4.2.1 Diversificação

Os resultados de diversificação foram obtidos através da aplicação do algoritmo SAND com mutações de ponto único sobre a RNA original para geração de 10 novas redes diversificadas. Dentre as 10 novas redes, o algoritmo alcançou aumento de diversidade entre 10,8 e 14,85 pontos percentuais na camada oculta, e alcançou melhorias de diversidade entre 74,69 e 75,16 pontos percentuais na camada de saída. Vale lembrar que a camada oculta era composta por 25 neurônios e a camada de saída era composta por 2 neurônios. Sendo assim, o que justifica a diferença de diversidade da RNA original para as RNAs diversificadas é o número de neurônios de cada camada, ou seja, o número de indivíduos de cada população. A população da camada oculta era composta de 25 neurônios, enquanto a população da camada de saída era composta por apenas 2 neurônios, logo a diversidade inicial dos indivíduos da camada oculta é consideravelmente melhor que a diversidade da camada de saída. Sendo assim, enquanto a diversidade na camada oculta apresentou um aumento de 78,78% para 93,63%, a diversidade na camada de saída apresentou um aumento de 24,35% para 99,51%, ou seja, quanto menor a população, mais eficaz é o processo de diversificação.

Para facilitar o entendimento, a Tabela 4 e o Gráfico 4 apresentam os resultados obtidos durante o processo de diversificação. Ambos apresentam a rede original (RNA Origem) e as 10 novas redes (RNA número da rede) com seus respectivos percentuais de diversidade por camada.

Como pode ser visualizado na Tabela 4, a melhor diversidade alcançada foi a diversidade da RNA 1, que apresentou um aumento de diversidade de 14,85 pontos percentuais na camada oculta e 74,76 pontos percentuais na camada de saída. Vale lembrar que os parâmetros utilizados como condições de parada são apresentados na seção 3.1.1.4, sendo 5000 o número máximo de gerações e 99,5% o percentual máximo de diversidade.

Tabela 4 - Diversidade das redes por camada com mutação em múltiplos pontos

RNA	Camada oculta	Camada de saída
Origem	78,78%	24,35%
1	93,63%	99,11%
2	90,75%	99,51%
3	89,70%	99,32%
4	89,64%	99,50%
5	93,57%	99,50%
6	90,42%	99,51%
7	90,99%	99,50%
8	89,58%	99,51%
9	92,54%	99,04%
10	93,01%	99,50%

Considerando esses parâmetros e a camada oculta, se pode concluir que o algoritmo não alcançou o máximo de diversidade em nenhuma rede, ou seja, o critério de parada utilizado foi o máximo de gerações (Tabela 4). Considerando a camada saída, se pode concluir que o algoritmo conseguiu alcançar o máximo de diversidade na maioria das redes, apenas três redes RNA 1, RNA 3 e RNA 9 não alcançaram (Tabela 4).

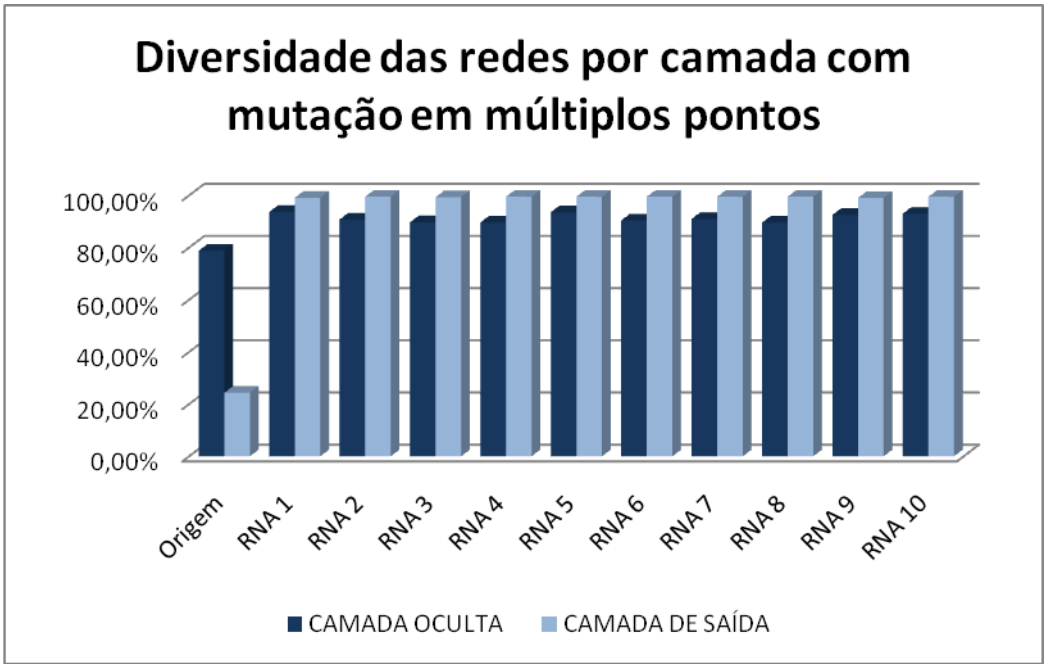

Gráfico 4 - Diversidade das redes por camada com mutação em múltiplos pontos

4.2.2 Treinamento

Os resultados de treinamento foram obtidos através da aplicação das RNAs sobre a base KDDTrain+_20Percent, que representa 20% da base KDDTrain+ que contem dados de treinamento, apresentada na seção 2.8.

Após o treinamento das redes sobre a base a KDDTrain+_20Percent, as mesmas eram testadas sobre a base KDDTest+ com o intuito de obter o MSE (*Mean Square Error*), ou seja, o erro médio quadrado da rede (FANN, 2012).

Dentre as 10 novas redes, o algoritmo em nenhum caso conseguiu reduzir o número de épocas de treinamento e em metade dos casos, ou seja, em cinco casos conseguiu reduzir o MSE da rede original.

Para facilitar o entendimento, a Tabela 5 e o Gráfico 5 apresentam os resultados obtidos durante o processo de treinamento. Ambos apresentam a rede original (RNA Origem) e as 10 novas redes (RNA número da rede) com seus respectivos valores de MSE e números de épocas de treinamento.

Tabela 5 - Treinamento das redes de múltiplos pontos sobre base KDDTrain+_20Percent

RNA	Épocas	MSE
Origem	200	188×10^{-3}
1	200	169×10^{-3}
2	200	166×10^{-3}
3	200	221×10^{-3}
4	200	204×10^{-3}
5	200	203×10^{-3}
6	200	183×10^{-3}
7	200	204×10^{-3}
8	200	183×10^{-3}
9	200	154×10^{-3}
10	200	220×10^{-3}

Como pode ser visualizado na Tabela 5, o melhor valor de MSE alcançado foi o valor da RNA 9, que reduziu em 0,034 o MSE da RNA original; seguido pela RNA 2, que reduziu em 0,022 o MSE da RNA original. Vale lembrar que os parâmetros utilizados para treinamento são apresentados na seção 3.1.2, sendo 0,000001 a margem de erro desejada, 200 o número máximo de épocas de treinamento, e 10 o número de iterações entre os relatórios de treinamento.

Considerando esses parâmetros, se pode concluir que o algoritmo não conseguiu reduzir o número de épocas de treinamento em nenhuma rede. Quanto a MSE, o algoritmo conseguiu reduzir a margem em metade dos casos, ou seja, em cinco casos, redes RNA 1, RNA 2, RNA 6, RNA 8 e RNA 9 (Tabela 5 e Gráfico 5). A partir dos dados apontados pela Tabela 5 e Gráfico 5, se pode notar que o algoritmo não apresentou melhorias para o número de épocas do processo de treinamento, mas apresentou melhorias significativas em cinco casos para o MSE.

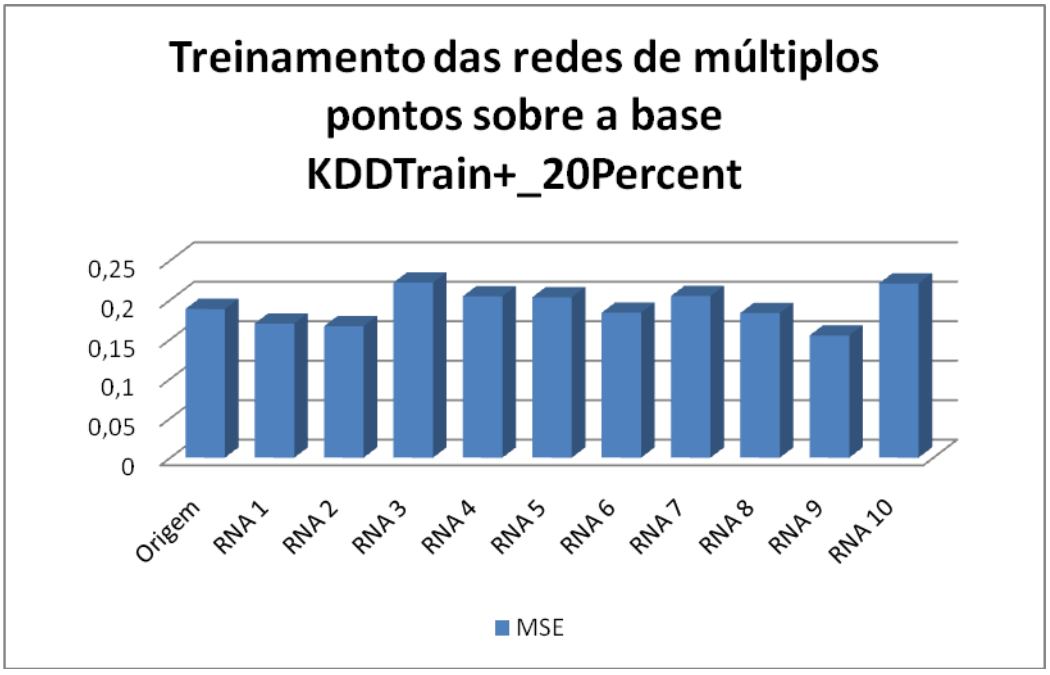

Gráfico 5 - Treinamento das redes de múltiplos pontos sobre a base KDDTrain+_20Percent

4.2.3 Execução

Os resultados de execução foram obtidos através da aplicação das RNAs sobre as 3 bases da NSL-KDD apresentadas na seção 2.8. Sendo elas:

- KDDTrain+: base que contem os dados de treinamento;
- KDDTrain+_20Percent: base que contem 20% dos dados da base KDDTrain+;
- KDDTest+: base que contem os dados para teste; e
- KDDTest-21: base que contem os dados para teste que não foram bem classificados.

Dentre as 10 novas redes, 5 redes apresentaram resultados positivos quanto a taxa de acerto. Apresentando melhorias entre 2% e 8% na base KDDTest-21, e melhorias entre 2% e 4% na base KDDTest+. As demais redes apresentaram resultados negativos quanto à taxa de acerto. Apresentando piora entre 2% e 4% na base KDDTest-21, e piora entre 1% e 2% na base KDDTest+. Vale lembrar que a base KDDTrain+_20Percent representa 20% da base KDDTrain+, e foi utilizada durante o processo de treinamento, o que justifica a diferença significativa em taxa de acerto, alcançado resultados entre 99% e 100% de taxa de acerto.

Para facilitar o entendimento, a Tabela 6 e o Gráfico 6 apresentam os resultados obtidos durante o processo de execução. Ambos apresentam a rede original (RNA original) e as 10 novas redes (RNA número da rede) com suas respectivas taxas de acerto por base de dados.

Como pode ser visualizado na Tabela 6, a melhor rede encontrada foi a RNA 9 que apresentou 8 pontos percentuais de melhoria na taxa de acerto na base KDDTest-21 e 4 pontos percentuais de melhoria na taxa de acerto na base KDDTest+; seguida pela RNA 2, que apresentou 6 pontos percentuais de melhoria na taxa de acerto na base KDDTest-21 e 4 pontos percentuais de melhoria na taxa de acerto na base KDDTest+. A pior rede encontrada foi a RNA 10 que apresentou 4 pontos percentuais de piora na taxa de acerto na base KDDTest-21 e 2 pontos percentuais de piora na taxa de acerto na base KDDTest+; seguida pela RNA 3 que apresentou 3 pontos percentuais de piora na taxa de acerto na base KDDTest-21 e 1 ponto percentual de piora na taxa de acerto na base KDDTest+.

Tabela 6 - Execução das redes de múltiplos pontos sobre as bases de dados

RNA	KDDTest-21	KDDTest+	KDDTrain+_20Percent	KDDTrain+
original	59,00%	78,00%	99,00%	99,00%
1	64,00%	81,00%	100,00%	99,00%
2	65,00%	82,00%	99,00%	99,00%
3	56,00%	77,00%	99,00%	99,00%
4	57,00%	78,00%	99,00%	99,00%
5	57,00%	78,00%	99,00%	99,00%
6	61,00%	80,00%	99,00%	99,00%
7	57,00%	77,00%	99,00%	99,00%
8	63,00%	80,00%	99,00%	99,00%
9	67,00%	82,00%	99,00%	99,00%
10	55,00%	76,00%	99,00%	99,00%

Considerando as taxas de acerto apresentadas pela Tabela 6 e pelo Gráfico 6, se pode concluir que o algoritmo consegue melhorar a taxa de acerto em todas as bases, mas não garante essa melhoria, uma vez que apresentou 8 e 4 pontos percentuais de melhoria nas bases KDDTest-21 e KDDTest+ com a RNA 9; e apresentou 4 e 2 pontos percentuais de piora nas bases KDDTest-21 e KDDTest+ com a RNA 10.

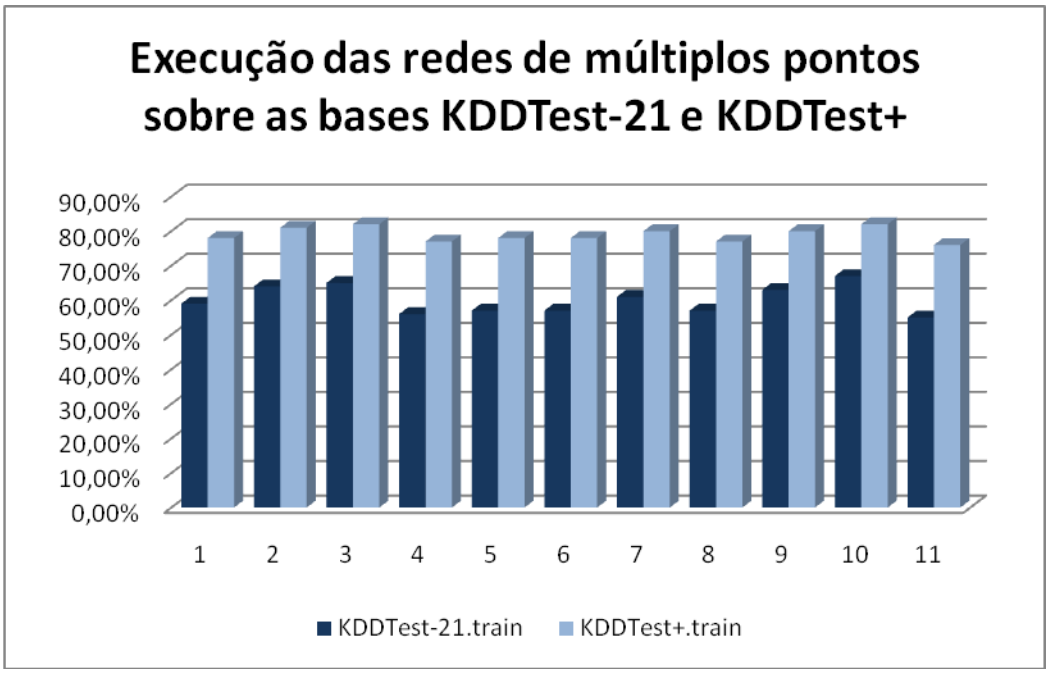

Gráfico 6 - Execução das redes de múltiplos pontos sobre as bases KDDTest-21 e KDDTest+

4.2.4 Análise geral

O processo de diversificação alcançou melhorias significativas quanto à diversidade das populações, apresentando aumento de diversidade entre 10,8 e 14,85 pontos percentuais na camada oculta e aumento de diversidade entre 74,69 e 75,16 pontos percentuais na camada de saída. Porém qualquer conclusão quanto à importância de utilização do algoritmo SAND considerando apenas esse processo é precipitada, uma vez que ainda não são conhecidos os resultados de treinamento e execução. Dessa forma, o que se pode concluir até esse ponto é que o algoritmo de fato melhora a diversidade.

O processo de treinamento não alcançou melhorias quanto ao número de épocas de treinamento, uma vez que em nenhum caso apresentou redução das épocas. Quanto a MSE o algoritmo alcançou melhorias significativas, uma vez que em metade dos casos, ou seja, em cinco casos apresentou melhorias entre 0,005 e 0,034. Porém qualquer conclusão quanto à importância de utilização do algoritmo SAND até esse processo ainda é precipitada, uma vez que ainda não são

conhecidos os resultados de execução. Dessa forma, o que se pode concluir até esse ponto é que o algoritmo de fato melhora a diversidade, e pode melhorar o processo de treinamento, ou seja, aumento de diversificação não implica necessariamente em melhoria no processo de treinamento.

O processo de execução foi o último sobre as bases de dados. Nesse processo foram encontrados resultados bem interessantes, nos quais o melhor resultado apresentado foi melhoria de 8 pontos percentuais de taxa de acerto na base KDDTest-21 e melhoria de 4 pontos percentuais de taxa de acerto na base KDDTest+. Nesse ponto, se pode concluir que o algoritmo de fato melhora a diversidade, pode melhorar o processo de treinamento e pode melhorar taxa de acerto, mas não garante melhorias, ou seja, aumento de diversificação não implica necessariamente em melhoria no processo de treinamento e melhoria nas taxas de acerto. A comparação dos resultados de taxas de acerto (Tabela 6) com os resultados de diversificação (Tabela 4) gera certo impasse, uma vez que as duas melhores redes encontradas RNA 9 e RNA 2 apresentam percentual de diversidade próximo ao percentual de diversidade da RNA 10, que apresentou baixa melhoria no processo de execução.

Enquanto a RNA 9 apresenta 92,54% de diversidade (Tabela 4) e 67% de taxa de acerto na base KDDTest-21 (Tabela 6), a RNA 10 apresenta 93,01% de diversidade (Tabela 4) e 55% de taxa de acerto na mesma base. A princípio essa diferença de diversidade pode colocar em dúvida a importância do processo de diversificação, mas analisando os indivíduos da população da camada oculta da RNA 9 (Figura), da RNA 2 (Figura) e da RNA 10 é possível visualizar semelhanças e diferenças quanto a disposição dos indivíduos.

Em ambas as figuras (Figura 9 e 10) é possível notar uma concentração considerável de indivíduos entre o segundo e o quarto quadrante do plano cartesiano. Dessa forma é possível encontrar um padrão entre as duas redes (RNA 9 e RNA 2), existe um número considerável de indivíduos em determinada área, apontando assim uma região promissora no espaço de buscas, uma vez que as duas redes apresentaram os melhores resultados de MSE (Tabela 5) e taxas de acerto (Tabela 6); e o mesmo não acontece para a RNA 10 (Figura 11).

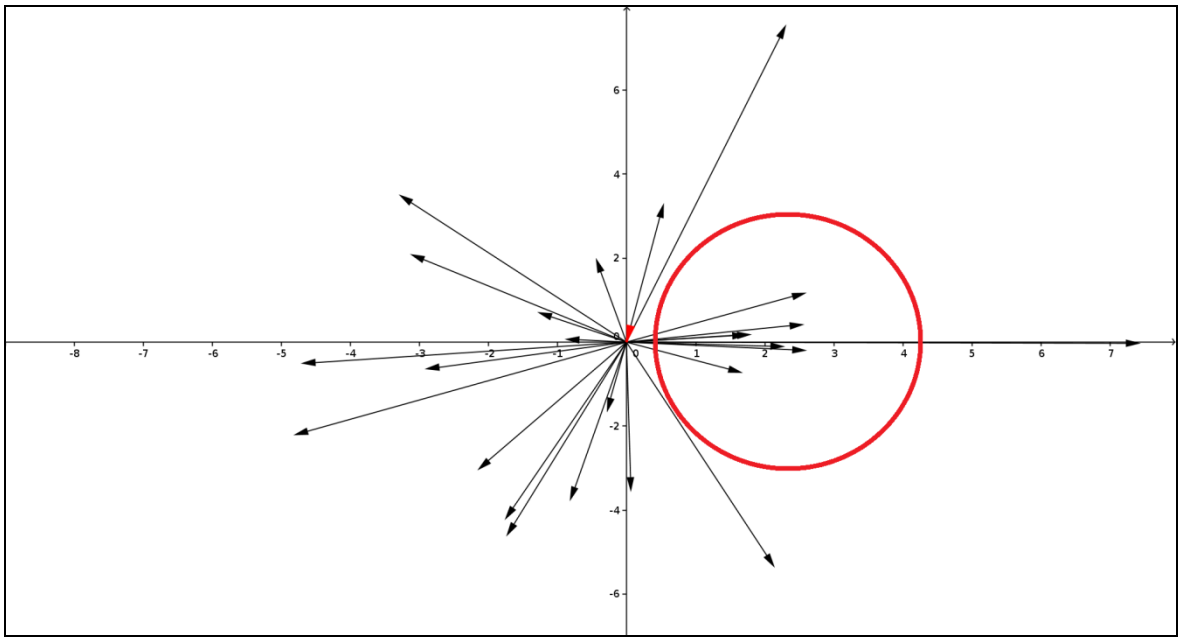

Figura 9 - População da camada oculta da RNA 9 utilizando mutação em múltiplos pontos

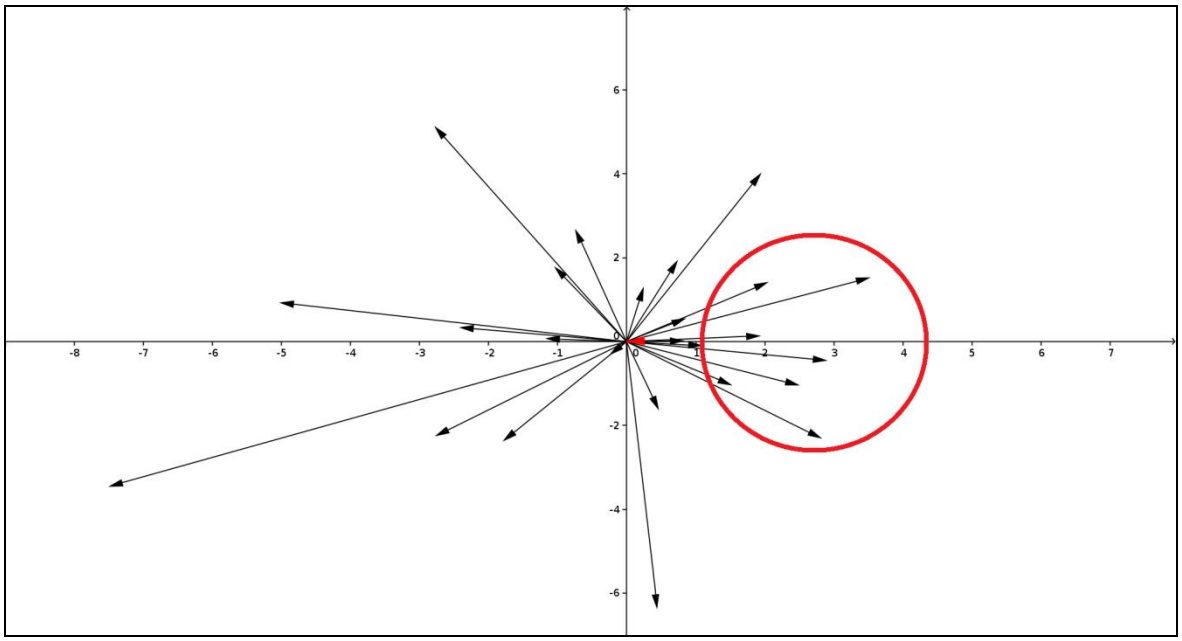

Figura 10 - População da camada oculta da RNA 2 utilizando mutação em múltiplos pontos

A utilização do algoritmo com mutações em múltiplos pontos é muito interessante. As melhorias no processo de treinamento foram satisfatórias, mesmo não conseguindo reduzir número de épocas, metade dos casos conseguiram reduzir a taxa de MSE. Quanto ao processo de execução, qualquer melhoria nas taxas de acerto é bem vinda por menor que seja devido às deficiências apontadas nos SDIs e a necessidade constante de melhoria citadas anteriormente, e melhorias de 8 pontos

percentuais de taxa de acerto na base KDDTest-21 e 4 pontos percentuais de taxa de acerto na base KDDTest+ foram alcançadas.

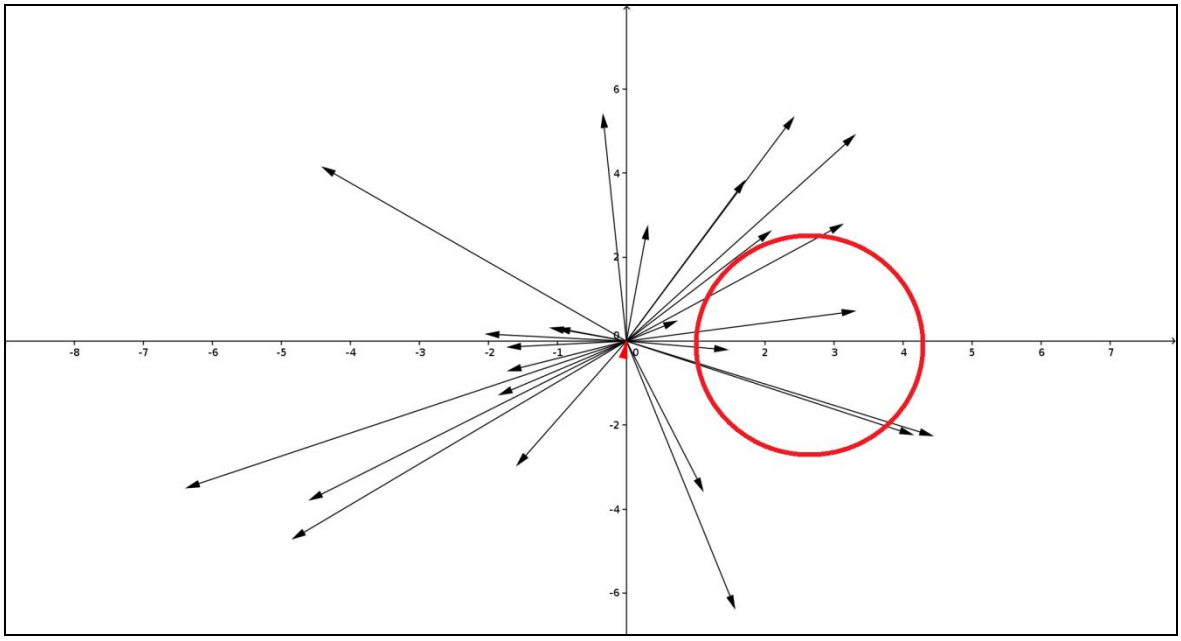

Figura 11 - População da camada oculta da RNA 10 utilizando mutação em múltiplos pontos

4.3 MUTAÇÃO EM PONTO ÚNICO E MUTAÇÃO EM MÚLTIPLOS PONTOS

Analisando os resultados obtidos com mutação de ponto único e mutação em múltiplos pontos é possível chegar a conclusões interessantes. A análise é apresentada seguindo a estrutura utilizada anteriormente para cada tipo de mutação, começando pela diversificação, depois pelo treinamento e execução, e ao final um resumo.

4.3.1 Diversificação

Quanto ao processo de diversificação ambos os tipos de mutação apresentaram taxas parecidas. O algoritmo utilizando mutação de ponto único em um caso conseguiu gerar uma RNA com 97,96% de diversidade, apresentando uma taxa muito acima da média das demais redes. Já o algoritmo utilizando mutação em múltiplos pontos não conseguiu gerar nenhuma rede que apresentasse uma taxa próxima de 97,96%, seu melhor caso resultou em 93,63% de diversidade. Porém enquanto o algoritmo utilizando mutação de ponto único conseguiu alcançar o máximo de diversidade em apenas dois casos na camada de saída, o algoritmo utilizando mutação em múltiplos pontos conseguiu alcançar o máximo de diversidade em sete casos.

Dessa forma o algoritmo com mutação em múltiplos pontos se apresenta mais eficiente devido aos percentuais de diversidade obtidos na camada de saída. O fato do algoritmo com mutação em ponto único ter encontrado uma rede com 97,96% de diversidade não é suficiente para poder ser considerado mais eficaz, uma vez que valores aleatórios são utilizados durante o processo, e apenas uma rede apresentou uma taxa tão elevada.

4.3.2 Treinamento

Quanto ao processo de treinamento ambos os tipos de mutação não conseguiram reduzir o número de épocas de treinamento. Em todos os casos foram necessárias 200 épocas. O que difere os dois tipos de mutação foram as taxas de MSE encontradas. Enquanto o algoritmo com mutação em ponto único conseguiu reduzir a taxa em apenas dois casos, com melhorias entre 0,001 e 0,004; o algoritmo com mutação em múltiplos pontos conseguiu reduzir a taxa em cinco casos, com melhorias entre 0,005 e 0,034. Dessa forma o algoritmo com mutação

em múltiplos pontos se apresenta mais eficiente devido às melhorias obtidas nas taxas de MSE.

4.3.3 Execução

Quanto ao processo de execução ambos os tipos de mutação apresentaram melhorias nas taxas de acerto. O que difere os dois tipos de mutação foram o número de casos de melhoria de taxas de acerto e as variações encontradas nessas taxas. Enquanto o algoritmo com mutação em ponto único conseguiu melhorias de taxa de acerto em 4 casos, com variações de 1 a 3 pontos percentuais; o algoritmo com mutação em múltiplos pontos conseguiu melhorias em 5 casos, com variações de 2 a 8 pontos percentuais.

Dessa forma o algoritmo com mutação em múltiplos pontos se apresenta mais eficiente devido ao fato de apresentar maior número de soluções positivas e maior variação de melhoria de taxa de acerto, apresentando no melhor caso um aumento de 8 pontos percentuais, enquanto o melhor caso do algoritmo de ponto único aumentou 3 pontos percentuais.

4.3.4 Análise geral

A partir das análises realizadas nos tópicos anteriores, é possível concluir que a utilização de mutação em múltiplos pontos se apresenta mais interessante. O único ponto em que o algoritmo com mutação em múltiplos pontos apresentou resultado inferior ao algoritmo com mutação em ponto único foi quanto à diversificação, no qual o algoritmo de ponto único alcançou 97,96% de diversidade em determinado caso. Porém como citado anteriormente, esse caso não é suficiente para considerar mutação de ponto único como mais eficaz, uma vez que valores

aleatórios são utilizados durante o processo, apenas uma rede apresentou uma taxa tão elevada e os resultados obtidos nos processos de treinamento e execução são melhores para a mutação em múltiplos pontos.

5 CONCLUSÃO

Chegou-se a conclusão que a utilização do algoritmo SAND para inicialização de pesos de Redes Neurais Artificiais pode gerar resultados muito interessantes como redução de Erro Médio Quadrático (*MSE - Mean Square Error*) e aumento significativo de taxas de acertos.

De acordo com os testes realizados e com os resultados obtidos conclui-se que a utilização do algoritmo SAND é promissora principalmente com mutação em múltiplos pontos, uma vez que o ajuste de pesos iniciais pode resultar em melhorias significativas para a Rede Neural Artificial como melhorias de taxa de acerto de 3 e 8 pontos percentuais na base KDDTest-21 e 2 e 4 pontos percentuais na base KDDTest+.

Outro ponto que se pôde concluir através dos testes é que a diversificação de repertório não implica necessariamente em melhorias, uma vez que em alguns testes, apesar de altas taxas de diversidade serem alcançadas, os resultados não foram positivos. Sendo necessário realizar vários testes com diferentes tipos de mutação. Dessa forma, analisar o processo de diversificação considerando apenas o percentual de diversidade não é suficiente, pois um caso com menor percentual de diversidade também pode levar os indivíduos para mais perto do ótimo global.

Com a realização de vários testes com diferentes tipos de mutação é possível gerar resultados muito positivos. É possível aumentar as taxas de acerto, o que resultaria em maior eficiência e confiabilidade para ferramentas como o Sistema de Detecção de Intrusão, o que seria muito interessante devido à necessidade constante de melhorias nessas ferramentas para aumento da segurança computacional.

6 TRABALHOS FUTUROS

Sugestão de trabalhos futuros poderia ser a utilização de novos conceitos de mutação. Como mutações mais agressivas, tanto no sentido do percentual de genes para mutação quanto para as taxas de mutação em si.

Outra sugestão seria utilização de mais técnicas de Inteligência Artificial, através do levantamento das deficiências da técnica apresentada, e a busca de técnicas que consigam suprir essas deficiências.

Seria interessante também o desenvolvimento de um Sistema de Detecção de Intrusão utilizando as técnicas e conceitos apresentados. A utilização de outras bases de testes também seria interessante para avaliação da implementação antes de partir direto para a aplicação das técnicas e conceitos sobre um Sistema de Detecção de Intrusão.

REFERÊNCIAS

BRITT, Winard *et al*. **Computer Defense using Artificial Intelligence.** In: Proceedings of the 2007 Spring Simulation Multiconference, Volume 3, Norfolk, Virginia, Março, 2007. Spring Simulation Multiconference. Society for Computer Simulation International, San Diego, CA, 2007.

CERT.BR. **Cartilha de segurança para Internet.** Disponível em: <http://cartilha.cert.br>. Acesso em: 01 jun. 2012.

COSTA, Nilson Santos. **Proteção de sistemas elétricos considerando aspectos de segurança da Rede de Comunicação.** Tese (Doutorado). Curso de Engenharia Elétrica, Escola de Engenharia de São Carlos, São Carlos, 2007.

DE CASTRO, L. N. **Engenharia Imunológica:** desenvolvimento e aplicação de ferramentas computacionais inspiradas em Sistemas Imunológicos Artificiais. Tese (Doutorado). Faculdade de Engenharia Elétrica e de Computação, Universidade Estadual de Campinas, Campinas, 2001.

DE CASTRO, L. N.; VON ZUBEN, F. J. **Artificial Immune Systems:** part I – basic theory and applications, Technical Report – RT DCA 01/99, p. 95, 1999.

FANN. **Fast Artificial Neural Network Library.** Disponível em: <http://sourceforge.net/projects/fann/>. Acesso em: 26 dez. 2012.

FARMER, J. D.; PACKARD, N.; PERELSON, A. **The immune system, adaptation and machine learning.** Physica D, v. 22, p. 187–204, 1986.

KDDCUP99. **KDD Cup 1999 Data.** Disponível em: <http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html>. Criada em: 1999. Acesso em: 03 dez. 2012.

MACHADO, Renato Bobsin. **Uma abordagem de Detecção de Intrusão baseada em Sistemas Imunológicos Artificiais e Agentes Móveis.** Dissertação (Mestrado) – Universidade Federal de Santa Catarina, Florianópolis, 2005.

NSL-KDD. **The NSL-KDD Data Set**. Disponível em: <http://www.iscx.ca/NSL-KDD/>. Criada em: 2009. Acesso em: 02 dez. 2012.

PYTHON. **Python Programming Language.** Disponível em: <http://www.python.org>. Acesso em: 26 dez. 2012.

SILVA, Jacson R. C. **Sistemas de Detecção de Intrusão com técnicas de Inteligência Artificial.** Dissertação (Mestrado) – Universidade Federal de Viçosa, Viçosa, 2011.

TAVALLAEE, Mahbod *et al*. **A detailed analysis of the KDD CUP 99 Data Set**. Proceeding of the 2009 IEEE Symposium on Computational Intelligence in Security and Defense Applications. (CISDA), 2009.

UCHÔA, Joaquim Q. **Algoritmos Imunoinspirados aplicados em Segurança Computacional:** Utilização de Algoritmos Inspirados no Sistema Imune para Detecção de Intrusos em Redes de Computadores. Tese (Doutorado) – Universidade Federal de Minas Gerais, Belo Horizonte, 2009.

WANG, Jie. **Computer Network Security:** theory and practice. Higher Education Press, Beijing and Springer-Verlag GmbH Berlin Heidelberg, 2009.

WU, Shelly Xiaonan; BANZHAF, Wolfang. T**he use of Computational Intelligence in Intrusion Detection Systems:** a review. Applied Soft Computing, Volume 10, Janeiro 2010, pág.1-35.

ANEXOS

ANEXO I – Biblioteca SAND

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-
'''
ANEXO I - Biblioteca SAND
@author: Vinícius Campista Brum <viniciuscampistabrum@gmail.com>
Adaptado de De Castro (2001)
'''

from math   import exp
from math   import sqrt
from random import randint
from random import random

def gera( N,L ):
    '''
    Método que gera uma população inicial de anticorpos.

    @param N: número de anticorpos
    @param L: comprimento dos anticorpos
    @return população de N anticorpos de comprimento L
    '''
    Ab = [ list() ] * N

    for i in range( N ):
        norma_Abi = 0

        while norma_Abi == 0:
            Ab[ i ] = list()
            soma_quadrados_Abi = 0

            for j in range( L ):
                Abi_k = randint( -1,1 ) * random() / 10
                soma_quadrados_Abi += Abi_k ** 2
                Ab[ i ].append( Abi_k )

            norma_Abi = sqrt( soma_quadrados_Abi )

    return Ab

def distancia_euclidiana( Abi,Abj ):
    '''
    Método que calcula distância euclidiana entre dois anticorpos

    @param Abi: anticorpo i
    @param Abj: anticorpo j
    @return distância euclidiana entre Abi e Abj
```

```
        '''
        distancia_Abi_Abj   = 0
        distancia_Abi_k_Abj_k = 0

        for Abi_k,Abj_k in zip( Abi,Abj ):
            distancia_Abi_k_Abj_k += ( Abi_k - Abj_k ) ** 2

        distancia_Abi_Abj = sqrt( distancia_Abi_k_Abj_k )

        return distancia_Abi_Abj

def vetor_unitario( Abi ):
        '''
        Método que calcula o vetor unitário de determinado anticorpo

        @param Abi: anticorpo i
        @return vetor unitário de Abi
        '''
        soma_quadrados_Abi = 0
        norma_Abi       = 0

        for Abi_k in Abi:
            soma_quadrados_Abi += Abi_k ** 2

        norma_Abi = sqrt( soma_quadrados_Abi ) if soma_quadrados_Abi else 1

        uAbi = [ ( Abi_k / norma_Abi ) for Abi_k in Abi ]

        return uAbi

def vetor_direcional_medio( Ab ):
        '''
        Método que calcula o vetor direcional médio de determinada população de anticorpos

        @param Ab: população de anticorpos
        @return vetor direcional medio de Ab
        '''
        mAb = [ 0 ] * len( Ab[ 0 ] )

        for Abi in Ab:
            uAbi = vetor_unitario( Abi )

            for index,uAbi_k in enumerate( uAbi ):
                mAb[ index ] += uAbi_k

        mAb = [ ( mAb_k / len( Ab ) ) for mAb_k in mAb ]

        return mAb

def distancia_vetor_medio_origem( mAb ):
        '''
        Método que calcula a distância entre o vetor direcional médio de determinada população e a origem do
sistema de coordenadas

        @param mAb: vetor direcional médio de determinada população de anticorpos
        @return distância entre mAb e a origem do sistema de coordenadas
```

```
        '''
        distancia_mAb_origem = 0

        for mAb_k in mAb:
            distancia_mAb_origem += mAb_k ** 2

        distancia_mAb_origem = sqrt( distancia_mAb_origem )

        return distancia_mAb_origem

def medida_percentual_diversidade_populacao( distancia_mAb_origem ):
        '''
        Método que calcula o percentual de diversidade de determinada população de anticorpos

        @param distancia_mAb_origem: distância entre o vetor direcional médio de determinada população de
anticorpos e a origem do sistema de coordenadas
        @return percentual de diversidade de determinada populacao
        '''
        medida_percentual_diversidade = 100 * ( 1 - distancia_mAb_origem )

        return medida_percentual_diversidade

def energy( Ab ):
        '''
        Método que calcula a energia de determinada população

        @param Ab: população de anticorpos
        @return energia de Ab
        '''
        soma_distancias_Ab = 0

        for Abi in Ab:
            for Abj in Ab[ ( Ab.index( Abi ) + 1 ): ]:
                soma_distancias_Ab += distancia_euclidiana( Abi,Abj )

        return soma_distancias_Ab

def hypermut( Ab ):
        '''
        Método que gera uma perturbação em determinda população de anticorpos

        @param Ab: população de anticorpos
        @return população gerada à partir de uma pertubação em Ab
        '''
        distancia_Abi_Abj  = 0
        menor_distancia_Ab = 0

        Abt = list()

        pares_indices_menor_distancia_Ab       = list()
        par_indices_menor_distancia_Ab_mutacao = list()
        indice_Abi_mutacao      = 0

        index_i = index_j = 0

        for index_i,Abi in enumerate( Ab ):
```

```
            Abt.append( [ Abi_k for Abi_k in Abi ] )
            index_j = index_i + 1

            for Abj in Ab[ ( index_i + 1 ): ]:
                distancia_Abi_Abj = distancia_euclidiana( Abi,Abj )

                if index_i == 0 and index_j == 1:
                    menor_distancia_Ab = distancia_Abi_Abj
                    pares_indices_menor_distancia_Ab.append( [ index_i,index_j ] )
                elif distancia_Abi_Abj < menor_distancia_Ab:
                    menor_distancia_Ab              = distancia_Abi_Abj
                    pares_indices_menor_distancia_Ab = list()
                    pares_indices_menor_distancia_Ab.append( [ index_i,index_j ] )
                elif distancia_Abi_Abj == menor_distancia_Ab:
                    pares_indices_menor_distancia_Ab.append( [ index_i,index_j ] )

                index_j += 1

        par_indices_menor_distancia_Ab_mutacao = pares_indices_menor_distancia_Ab[ randint( 0,( len(
pares_indices_menor_distancia_Ab ) - 1 ) ) ]
        indice_Abi_mutacao                = par_indices_menor_distancia_Ab_mutacao[ randint( 0,1 ) ]
        gene_indice_Abi_mutacao             = randint( 0,( len( Abt[ indice_Abi_mutacao ] ) - 1 ) )
        taxa_mutacao                    = randint( -1,1 ) * random()

        Abt[ indice_Abi_mutacao ][ gene_indice_Abi_mutacao ] += taxa_mutacao

        return Abt

def sand( gen,beta,delta,diversidade_maxima,Ab=False,N=False,L=False ):
        '''
        Método que busca otimizar a diversidade de determinada população de anticorpos

        @param gen: número de máximo de gerações
        @param beta:
        @param delta:
        @param diversidade_maxima:
        @param Ab: população de anticorpos
        @param N: número de anticorpos da população          ( utilizado para fins de testes... onde a população
inicial era gerada através da função gera )
        @param L: comprimento dos anticorpos da população      ( utilizado para fins de testes... onde a população
inicial era gerada através da função gera )
        @return tupla: número de gerações,população 'diversificada' gerada à partir de Ab
        '''
        t    = 0
        ct   = 0
        E    = 0
        T    = 1

        populacao_melhor_diversidade = list()
        taxa_melhor_diversidade     = 0
        energia_melhor_diversidade  = 0

        Ab = Ab if Ab else gera( N,L )

        if not Ab:
            print
```

```
            print 'População de anticorpos não definida!'
            print '\tVerifique se a população inicial ou as configurações para geração de população inicial foram informadas.'
            print
            return Ab

        while t < gen and taxa_melhor_diversidade < diversidade_maxima:
            E   = energy( Ab )
            Abt = hypermut( Ab )
            Et  = energy( Abt )
            dE  = E - Et

            mAb                 = vetor_direcional_medio( Ab )
            distancia_mAb_origem  = distancia_vetor_medio_origem( mAb )
            diversidade_Ab        = medida_percentual_diversidade_populacao( distancia_mAb_origem )
            mAbt                = vetor_direcional_medio( Abt )
            distancia_mAbt_origem = distancia_vetor_medio_origem( mAbt )
            diversidade_Abt       = medida_percentual_diversidade_populacao( distancia_mAbt_origem )
            variacao_diversidade  = diversidade_Ab - diversidade_Abt

            if t == 0:
                if diversidade_Ab > diversidade_Abt:
                    populacao_melhor_diversidade = [ [ Abi_k for Abi_k in Abi ] for Abi in Ab ]
                    taxa_melhor_diversidade      = diversidade_Ab
                    energia_melhor_diversidade   = E
                else:
                    populacao_melhor_diversidade = [ [ Abti_k for Abti_k in Abti ] for Abti in Abt ]
                    taxa_melhor_diversidade      = diversidade_Abt
                    energia_melhor_diversidade   = Et
            else:
                if diversidade_Abt > taxa_melhor_diversidade:
                    populacao_melhor_diversidade = [ [ Abti_k for Abti_k in Abti ] for Abti in Abt ]
                    taxa_melhor_diversidade      = diversidade_Abt
                    energia_melhor_diversidade   = Et
                elif diversidade_Abt == taxa_melhor_diversidade and Et < energia_melhor_diversidade:
                    populacao_melhor_diversidade = [ [ Abti_k for Abti_k in Abti ] for Abti in Abt ]
                    energia_melhor_diversidade   = Et

            if variacao_diversidade < 0:
                Ab = Abt
                E  = Et
                ct = 0
            elif variacao_diversidade > 0:
                num_random = random()
                num_exp    = exp( ( variacao_diversidade * -1 ) / T )

                if num_random < num_exp:
                    Ab = Abt
                    E  = Et
                    ct = 0
            else:
                ct = ct + 1

                if ct % delta == 0:
                    T  = beta * T
                    ct = 0
```

```
        t += 1

    return t,populacao_melhor_diversidade
```

ANEXO II – Criação da Rede Neural Artificial

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-
'''
ANEXO II - Criação da Rede Neural Artificial
@author: Vinícius Campista Brum <viniciuscampistabrum@gmail.com>
Adaptado de Silva (2011)
'''

import sys
from pyfann import libfann

nome_rna = ''

if len( sys.argv ) != 2:
	nome_rna = 'rna_original.net'
else:
	nome_rna = sys.argv[ 1 ]

taxaConexao     = 1
taxaAprendizado = 0.3
momento         = 0.3

neuroniosEntrada = 41
neuroniosOcultos = 25
neuroniosSaida   = 2

rna = libfann.neural_net()
rna.create_sparse_array( taxaConexao,( neuroniosEntrada,neuroniosOcultos,neuroniosSaida ) )
rna.set_learning_rate( taxaAprendizado )
rna.set_learning_momentum( momento )
rna.set_activation_function_hidden( libfann.SIGMOID_STEPWISE )
rna.set_activation_function_output( libfann.SIGMOID )
rna.set_training_algorithm( libfann.TRAIN_INCREMENTAL )

rna.save( nome_rna )
```

ANEXO III – Aplicação do SAND sobre a Rede Neural Artificial

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-
'''
ANEXO III - Aplicação do SAND sobre a Rede Neural Artificial
@author: Vinícius Campista Brum <viniciuscampistabrum@gmail.com>
Adaptado de De Castro (2001)
'''

import sys
from libsand import distancia_vetor_medio_origem
from libsand import medida_percentual_diversidade_populacao
from libsand import sand
from libsand import vetor_direcional_medio
from pyfann.neural_net import neural_net
from time import time

def get_pesos_conexoes_de( id_neuronio,conexoes ):
    pesos = list()

    for conexao in conexoes:
        if conexao.to_neuron == id_neuronio:
            pesos.append( conexao.weight )

    return pesos

nome_rna_original   = ''
nome_rna_modificada = ''

if len( sys.argv ) != 3:
    nome_rna_original   = '../rna_original.net'
    nome_rna_modificada = '../rna_modificada_01.net'
else:
    nome_rna_original   = sys.argv[ 1 ]
    nome_rna_modificada = sys.argv[ 2 ]

rna = neural_net()
rna.create_from_file( nome_rna_original )

camadas,conexoes = [],[]

rna.get_connection_array( conexoes )
rna.get_layer_array( camadas )

repertorio_inicial,repertorio_diversificado = [ [],[] ],[ [],[] ]

for id_neuronio in range( 42,67 ):
    repertorio_inicial[ 0 ].append( get_pesos_conexoes_de( id_neuronio,conexoes ) )

for id_neuronio in range( 68,70 ):
    repertorio_inicial[ 1 ].append( get_pesos_conexoes_de( id_neuronio,conexoes ) )
```

```
print '-' * 80
print '\tIniciando processo de diversificação...'
print '-' * 80 + '\n'

for idAb,Ab in enumerate( repertorio_inicial ):
	mAb                = vetor_direcional_medio( Ab )
	distancia_mAb_origem = distancia_vetor_medio_origem( mAb )
	diversidade_mAb      = medida_percentual_diversidade_populacao( distancia_mAb_origem )

	print 'camada',( idAb + 1 )
	print '\n\t',len( Ab ),'neurônios com',len( Ab[ 0 ] ),'sinapses\n'

	inicio = time()
	numero_geracoes,repertorio_diversificado[ idAb ] = sand( 5000,0.9,5,99.5,Ab )
	termino = time()

	mAbt                = vetor_direcional_medio( repertorio_diversificado[ idAb ] )
	distancia_mAbt_origem = distancia_vetor_medio_origem( mAbt )
	diversidade_mAbt      = medida_percentual_diversidade_populacao( distancia_mAbt_origem )

	print '\napós',round( ( termino - inicio ),2  ),'segundos foram testadas',numero_geracoes,'gerações...\n'

	print 'diversidade inicial.:',round( diversidade_mAb,2 ),'%'
	print 'diversidade final...:',round( diversidade_mAbt,2 ),'%'
	print 'resultado.....:',round( ( diversidade_mAbt - diversidade_mAb ),2 ),'% de melhoria'
	print '\n' + '-' * 80 + '\n'

for i,to_neuron in enumerate( range( 42,67 ) ):
	for j,from_neurom in enumerate( range( 0,42 ) ):
		rna.set_weight( from_neurom,to_neuron,repertorio_diversificado[ 0 ][ i ][ j ] )

for i,to_neuron in enumerate( range( 68,70 ) ):
	for j,from_neurom in enumerate( range( 42,68 ) ):
		rna.set_weight( from_neurom,to_neuron,repertorio_diversificado[ 1 ][ i ][ j ] )

rna.save( nome_rna_modificada )
```

ANEXO IV – Treinamento da Rede Neural Artificial

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-
"""

ANEXO IV - Treinamento da Rede Neural Artificial

@author: Vinícius Campista Brum <viniciuscampistabrum@gmail.com>
Adaptado de Silva (2011)
"""

import sys
from pyfann import libfann

nome_rna_original = ''
nome_rna_teste    = ''

if len( sys.argv ) != 3:
	nome_rna_original = 'rna_original.net'
	nome_rna_teste    = 'treina_' + nome_rna_original
else:
	nome_rna_original = sys.argv[ 1 ]
	nome_rna_teste    = sys.argv[ 2 ]

erroDesejado          = 0.000001
epocasTreino          = 200
iteracoesEntreRelatorios = 10
pathEntradas          = '/media/01CD837792BE62F0/Dados_TCC/Entradas/NSL-KDD/fann/'

rna = libfann.neural_net()
rna.create_from_file( nome_rna_original )

dadosTreinamento = libfann.training_data()
dadosTreinamento.read_train_from_file( pathEntradas+'normalizado-KDDTrain+_20Percent.train' )

print 'Treinando a Rede...'
rna.train_on_data( dadosTreinamento,epocasTreino,iteracoesEntreRelatorios,erroDesejado )

print 'Testando a Rede...'
dadosTeste = libfann.training_data()
dadosTeste.read_train_from_file( pathEntradas+'normalizado-KDDTest+.train' )

rna.reset_MSE()
rna.test_data( dadosTeste )
print 'MSE error on test data:',rna.get_MSE()

print 'Salvando a rede'
rna.save( nome_rna_teste )

rna.destroy()
```

ANEXO V – Execução da Rede Neural Artificial

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-
'''

ANEXO V - Execução da Rede Neural Artificial
@author: Vinícius Campista Brum <viniciuscampistabrum@gmail.com>
Adaptado de Silva (2011)
'''

from pyfann import libfann
from sys import argv

arquivosTeste = [
      '/media/01CD837792BE62F0/Dados_TCC/Entradas/NSL-KDD/fann/normalizado-KDDTest-21.train',
      '/media/01CD837792BE62F0/Dados_TCC/Entradas/NSL-KDD/fann/normalizado-KDDTest+.train',
      '/media/01CD837792BE62F0/Dados_TCC/Entradas/NSL-KDD/fann/normalizado-KDDTrain+_20Percent.train',
      '/media/01CD837792BE62F0/Dados_TCC/Entradas/NSL-KDD/fann/normalizado-KDDTrain+.train'
]

rna = libfann.neural_net()
rna.create_from_file( argv[ 1 ] )

print "Testando a Rede com as entradas..."
for filename in arquivosTeste:
   DadosTeste = libfann.training_data()
   DadosTeste.read_train_from_file( filename )

   entrada = DadosTeste.get_input()
   saida   = DadosTeste.get_output()

   Normal        = 0
   Normal_Ataque = 0
   Normal_Normal = 0
   Ataque        = 0
   Ataque_Normal = 0
   Ataque_Ataque = 0

   for i in range( len( entrada ) ):
      resposta = rna.run( entrada[ i ] )

      if saida[ i ][ 0 ] == 1.0:
         Ataque += 1

         if resposta[ 0 ] > resposta[ 1 ]:
            Ataque_Ataque += 1
         else:
            Ataque_Normal += 1
      else:
         Normal += 1

         if resposta[ 0 ] > resposta[ 1 ]:
```

```
            Normal_Ataque += 1
        else:
            Normal_Normal += 1

    DadosTeste.destroy_train()

    print "_" * 60
    print "[ Estatísticas %s ]" % ( filename.split( '/' )[ -1 ] )
    print "Total de entradas: ", Normal + Ataque

    try:
            print "Tráfego Malicioso Detectado (VP):     %d de %d %.0f%%" % ( Ataque_Ataque,Ataque,float(
Ataque_Ataque ) / float( Ataque ) * 100.0 )
    except:
            pass

    try:
            print "Tráfego Malicioso Não Detectado (FN): %d de %d %.0f%%" % ( Ataque_Normal,Ataque,float(
Ataque_Normal ) / float( Ataque ) * 100.0 )
    except:
            pass

    try:
            print "Tráfego Normal Detectado (VN):        %d de %d %.0f%%" % ( Normal_Normal,Normal,float(
Normal_Normal ) / float( Normal ) * 100.0 )
    except:
            pass

        try:
            print "Tráfego Normal Não Detectado (FP):    %d de %d %.0f%%" % ( Normal_Ataque,Normal,float(
Normal_Ataque ) / float( Normal ) * 100.0 )
    except:
            pass

        try:
            print "Taxa de Sucesso: %d de %d %.0f%%" % ( Ataque_Ataque+Normal_Normal, Normal+Ataque,( float(
Ataque_Ataque ) + float( Normal_Normal ) ) / float( Normal + Ataque ) * 100.0 )
    except:
            pass

    print      "_"*60

rna.destroy()
```