Reconhecimento de Intrusão em Redes de Computadores utilizando PyBrain

Heitor Scalco Neto
Universidade Federal de Lavras
Depart. de Ciência da Computação
Lavras, MG 37200-000
Email: heitorscalco@hotmail.com

Wilian Soares Lacerda
Universidade Federal de Lavras
Depart. de Ciência da Computação
Lavras, MG 37200-000
Email: lacerda@dcc.ufla.br

Victor Grudtner Boell
Universidade Federal de Lavras
Depart. de Ciência da Computação
Lavras, MG 37200-000
Email: grutnerv@gmail.com

***Resumo*—Este artigo apresenta uma abordagem para reconhecimento de intrusão em redes de computadores, utilizando técnicas de Redes Neurais Artificiais (RNA). Esta produção tem como principal objetivo aplicar a biblioteca *Pybrain* para classificação de tráfego de rede. A base de dados utilizada foi uma pequena porção (cerca de 10%) da KDDCup'99. Os resultados obtidos com a utilização da biblioteca *Pybrain* foram bastante satisfatórios, obteve-se uma média de acertos em torno de 98%, afirmando assim a viabilidade de utilizá-la para resolver problemas de classificação de tráfego de redes de computadores.**

I. Introdução

A comodidade e, em contrapartida, a dependência que a utilização da internet nos proporciona, tais como a facilidade para encontrar informações, as redes sociais e sistemas de gestão pessoal/empresarial online, faz com que sua utilização continue crescendo exponencialmente ao passar dos anos. A grande maioria da população mundial hoje, de alguma forma, depende da internet, tanto no ambiente empresarial quanto doméstico [1].

Anomalias de tráfego ocasionando falhas, principalmente negação de serviço, são a cada dia mais comuns. O padrão de identificar, diagnosticar e tratar essas anomalias em tempo real é de extrema importância para manter uma rede operando [2]. Visto isso, é necessário a utilização de um sistema que possa auxiliar na segurança desse volume de dados, não com o foco em nível doméstico, mas sim para a proteção de dados de ambiente corporativo e/ou acadêmico.

Em diferentes áreas de utilização da internet, são utilizados programas e sistemas operacionais que podem apresentar problemas de segurança. Para detectar e prevenir eventuais ocorrências dessas vulnerabilidades nos ambientes computacionais é necessário que um Sistema de Detecção de Intrusão em Redes (SDIR) seja utilizado [3]. Um SDIR é um conjunto de ferramentas de software que permite a análise e detecção de intrusões em redes de dados [4] [5]. Os sistemas detectores de intrusão em redes tem como característica oferecer um método para reduzir a possibilidade de intrusão, através da antecipação e acompanhamento dos ataques [6].

A proposta deste artigo é desenvolver um protótipo, aplicando a biblioteca *Pybrain* para detecção de intrusão em uma base de dados que é disponibilizada pela KDDCup'99 [29]. Ao final do artigo, o resultado esperado é utilizar uma rede neural devidamente treinada que é capaz de classificar distúrbios relacionados ao tráfego das redes de computadores, possibilitando o desenvolvimento de um protótipo de um sistema de detecção de intrusão em redes.

O restante deste artigo está organizado da seguinte forma: seção II apresenta uma breve revisão bibliográfica envolvendo Sistemas de detecção de intrusão em redes, Redes Neurais e a biblioteca *Pybrain*. Seção III descreve a metodologia utilizada para realizar os experimentos. Seção IV apresenta os resultados obtidos com a aplicação da biblioteca *Pybrain* para a resolução deste tipo de problema. Por fim, na Seção V são apresentadas as conclusões oriundas deste trabalho.

II. Revisão Bibliográfica

Esta Seção apresenta uma breve revisão bibliográfica sobre Sistemas de Detecção de Intrusão, Redes Neurais Artificiais e a biblioteca *Pybrain*.

A. Sistemas de Detecção de Intrusão em Redes

Intrusões em sistemas computacionais podem ser definidas como um conjunto de ações que objetivam comprometer a integridade, confidencialidade e/ou disponibilidade de um sistema ou rede. A detecção de intrusão permite identificar possíveis violações de políticas de segurança através do monitoramento de redes de computadores e/ou *hosts* específicos [8].

Um Sistema de detecção de Intrusão em Redes (SDIR) atua monitorando os pacotes que trafegam pela rede (Figura 1) e pode ser configurado de duas maneiras: por abuso ou por anomalia. A detecção por abuso é utilizada para detectar atividades maliciosas já conhecidas, de acordo com as políticas de segurança definidas na configuração inicial do SDIR. Este método fica limitado apenas aos ataques já definidos no SDIR. Por outro lado, a detecção por anomalias pode ser mais eficiente. O SDIR baseado em anomalias funciona analisando o tráfego e assumindo como padrão correto o uso normal da rede e/ou *host* [9].

Um dos maiores problemas nos sistemas operacionais é que alguns processos precisam de permissão de superusuário mas, muitas vezes, alguns processos dão mais permissão que o necessário. Se for possível obter a sequência das chamadas de sistema em uma execução normal, isso possibilita construir um mapa do comportamento normal, podendo assim, detectar um comportamento anômalo quando ocorrer uma intrusão [9].

Atualmente foram propostos alguns modelos de detecção por anomalias. São eles por estatística [10], por aprendizagem de máquina [11], por redes neurais artificiais [12], [13], [14], [15] e ainda por mineração de dados [16].

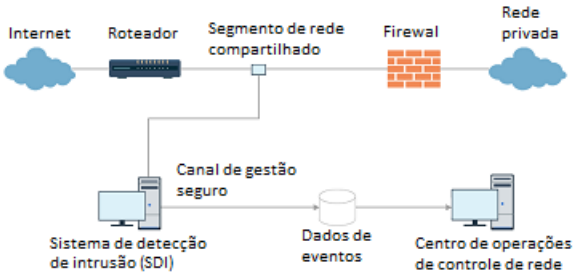

Figura 1. SDIR - Sistema de Detecção de Intrusão em Redes.

B. Redes Neurais Artificiais

O conceito de Redes Neurais Artificiais (RNAs) é definido por meio da utilização de técnicas computacionais desenvolvidas através de modelos matemáticos baseados no neurônio biológico e suas conexões no cerébro humano. A tentativa inicial de reproduzir o alto desempenho do cérebro humano em tarefas cognitivas extremamente complexas, motivou o desenvolvimento dos modelos de RNA [17]. Essa motivação originou-se pelo fato do cérebro possuir extrema capacidade de processamento, organização e principalmente generalização [18].

Em virtude do alto poder de generalização, redes neurais artificiais são utilizadas em diversas áreas e aplicações. Uma característica da rede neural é a capacidade de adaptação e aprendizado de acordo com a entrada de novos dados. Para que uma rede neural possa aprender, se faz necessário apresentar um conjunto de exemplos à mesma de forma sequencial e iterativa. Este processo é chamado de treinamento de redes neurais [7].

O modelo de neurônio mais simples é o perceptron, onde consiste de uma entrada, uma função de ativação e uma saída [28]. A rede neural do tipo perceptron de múltiplas camadas (conhecida como Multilayer Perceptron, MLP), é um conjunto de neurônios do tipo perceptron organizado em múltiplas camadas internas, além da entrada e saída, ao utilizar esta arquitetura é possível resolver problemas mais complexos variando o número de camadas e neurônios por camadas [28].

A eficiência e eficácia de uma rede neural se dá através do treinamento. Existem vários algoritmos para treinamento, este artigo utilizará o algoritmo de *backpropagation*. O *backpropagation* é uma regra de aprendizagem baseada na correção do erro pelo método do gradiente. Este algoritmo é composto por duas fases: *forward* (cálculo do erro) e *backward* (correção dos erros sinápticos). O treinamento opera em uma sequência de dois passos. São eles [19]:

- Um padrão é apresentado à camada de entrada da rede. A atividade resultante flui através da rede, camada por camada, até que a resposta seja produzida pela camada de saída.
- A saída obtida é comparada à saída desejada (pré-definida), caso não esteja correta, o erro é calculado. Esse erro é propagado a partir das camadas de saída até a camada de entrada, e os pesos das conexões das unidades das camadas internas vão sendo modificados conforme o erro é retropropagado (conceito de *back-propagation*).

De acordo com Lima et al. 2005 [18] o aprendizado supervisionado é um conjunto de dados que contém um conhecimento prévio do ambiente e são usados para o treinamento. Esse conhecimento é a combinação de entradas e suas saídas desejadas, dessa forma guiando o treinamento da rede. O conhecimento repassado e o erro resultante é avaliado, a partir disso é feito o ajuste dos pesos para a próxima época de treinamento. Como a resposta da rede é em função dos valores atuais do seu conjunto de pesos, esses são ajustados de forma a aproximar a saída da rede da saída desejada [20]. Uma técnica simples para fazer com que o treinamento da rede atinja o seu melhor nível é utilizar a taxa de momento. Essa taxa acrescenta efetivamente a inércia para o movimento do vetor peso e suaviza sua oscilação [21].

A função de ativação utilizada neste artigo foi a *Sigmoidal*. Essa função abrange valores entre 0 e 1. Deve-se evitar, através da normalização dos dados de entrada, que os valores atinjam os extremos, pois chega-se ao ponto de saturação.

C. Biblioteca Pybrain

Pybrain é uma biblioteca, em conjunto com a biblioteca científica *SciPy*, desenvolvida em Python, para facilitar aplicações envolvendo aprendizado de máquina. O *Pybrain* provê um *toolbox* para aprendizado supervisionado, não supervisionado, por reforço e outros. Os desenvolvedores enfatizaram a simplicidade, composicionalidade e a capacidade para combinar vários tipos de arquitetura e algoritmos de aprendizado de máquina. Apesar das limitações de desempenho do *python* (por ser uma linguagem interpretada), existem alguns padrões para otimização da performance na biblioteca que tornam o desempenho semelhante a linguagem C++. A biblioteca já foi citada por vários autores, entre eles: [22], [23], [24]. O *Pybrain* possui alguns requisitos [25], são eles: Scipy *(http://www.scipy.org)* e o Python 2.5 (ou versões mais atuais).

III. Metodologia

Analisar e classificar o tráfego de uma rede é sempre um desafio. A informação classificada em ”normal” ou ”intrusão” é essencial para o correto funcionamento de qualquer SDIR. A dificuldade em construir um SDIR eficiente é fazer com que o número de positivos reais seja grande, mas, em contra partida, o número de falsos positivos seja pequeno ou zero. Além disso, a extensa quantidade de informações que um pacote de rede possui pode dificultar a detecção. Muitas vezes, uma intrusão é caracterizada por uma ou mais variações nas informações de um pacote.

Para buscar uma solução para o problema descrito, é possível utilizar redes neurais artificiais, onde sua alta capacidade de generalização pode beneficiar a classificação de tráfego. Para auxiliar no desenvolvimento de uma rede neural completa, foi utilizado a biblioteca *Pybrain* e o algoritmo

de treinamento *backpropagation*. O processo de resolução do problema será descrito, com detalhes, nesta seção.

A. Base de Dados

A base de dados KDDCup'99 é, atualmente, a base de dados mais utilizada para avaliação e treinamento de sistemas de intrusão [26]. Essa base de dados foi preparada por M. Tavallaee [26] e foi construída baseada nas informações capturadas pela DARPA'98 IDS (*Defense Advanced Research Projects Agency* - http://www.darpa.mil/). Os dados contidos nesta base foram obtidos através da ferramenta de captura de pacotes *tcpdump*, durante 7 semanas, resultando em um arquivo de texto de aproximadamente 4 *gigabytes*, com aproximadamente 5 milhões de conexões obtidas, cada uma com cerca de 100 bytes. A base de dados KDDCup'99 possui 41 características para conexão e, cada conexão é caracterizada como "normal" ou um tipo de ataque [26]. Foi utilizado *10%* do total da base, porção na qual abrange todos os tipos de ataques disponíveis na base completa.

1) Tipos de intrusão: Os tipos de intrusão classificados mais frequentes apresentados pela base são:

- **Denial of Service Attack (DoS):** é um ataque em que o atacante faz com que os recursos de algum sistema fiquem extremamente ocupados com requisições inválidas, indisponibilizando o sistema. Exemplos de ataques de negação de serviço são: *apache, smurf, neptune, ping of death, back, mail bomb* e *UDP storm*;
- **User to Root Attack (U2R):** Também conhecido como elevação de privilégios, o atacante geralmente utiliza algum *exploit* para explorar uma vulnerabilidade em algum serviço. Através deste *exploit*, o atacante pode capturar a senha de *root* de um administrador e ganhar acesso privilegiado. Exemplos de ataques do tipo U2R: *buffer overflow, loadmodule, perl, rootkit*;
- **Remote to Local Attack (R2L):** É um ataque em que um usuário envia pacotes para uma máquina através de uma rede externa (*Internet*), onde o atacante não tem acesso e tenta explorar as vulnerabilidades como se fosse um usuário local. Estes ataques podem ser realizados através de *exploits*. Exemplos de ataques do tipo R2L: *guess password, imap, multihop, phf, spy, warez client, warez master*.
- **Probing Attack:** Consiste em fazer uma sondagem na rede, buscando o levantamento de todos os serviços que estão rodando em determinadas portas, desta forma o atacante pode saber qual *exploit* utilizar para o determinado alvo. Para esta varredura pode-se utilizar a ferramenta *nmap* (*https://nmap.org/*), *saint* (*http://www.saintcorporation.com/*), *portsweep* e *mscan*;

2) Características da Base: Pode-se dizer que esta base é dividida em dados descritos e não descritos, ou seja, ao final de cada pacote encontra-se um *tag* indicando se a informação caracteriza uma intrusão ou não. Cada pacote capturado consiste em 41 atributos diferentes e um valor alvo, sendo que o valor alvo descreve o tipo de ataque [27].

Existem vários recursos provenientes da base de dados, as tabelas I, II, III, apresentam as principais características obtidas.

TABELA I. CARACTERÍSTICAS BÁSICAS DAS CONEXÕES TCP INDIVIDUAIS.

Característica	Descrição	Tipo
Duration	tamanho (em segundos) da conexão	contínuo
Protocol	tipo do protocolo (Ex: TCP, UDP)	discreto
Service	serviço de rede no destino (Ex: http)	discreto
Src_bytes	*Bytes* de informação entre a origem e o destino	contínuo
Dst_bytes	*Bytes* de informação entre o destino e a origem	contínuo
Flag	Normal ou erro (Estado da conexão)	discreto
Land	1 se a conexão é de/para o mesmo *destino[tá itálico]*/porta, 0 o inverso	discreto
Wrong_fragment	Num. de fragmentos errados	contínuo
Urgent	Num. de pacotes urgentes	contínuo

TABELA II. RECURSOS DE CONTEÚDO DE CONEXÃO SUGERIDAS PELO CONHECIMENTO DO DOMÍNIO

Característica	Descrição	Tipo
hot	Número de indicadores *hot*	contínuo
num_failed_logins	Número de falhas de login	contínuo
logged_in	1 se está logado, 0 se não.	discreto
num_compromised	Número de condições comprometidas	contínuo
root_shell	1 se o privilégio de root está ativo, 0 se não.	discreto
su_attempted	1 se o comando "su root" foi tentado, 0 se não.	discreto
num_root	Número de acessos ao "root"	contínuo
num_file_creations	Número de operações de criação de arquivos	contínuo
num_shells	Número de prompts do shell abertos	contínuo
num_access_files	Número de operações de controle de acesso de arquivos	contínuo
num_outbound_cmds	Número de comandos de saída de uma sessão FTP	contínuo
is_hot_login	1 se o *login* provém da lista de *hot login*, 0 se não	discreto
is_guest_login	1 se o login é para convidado, 0 se não	discreto

TABELA III. CARACTERÍSTICAS DE TRÁFEGO USANDO UM *buffer* DE DOIS SEGUNDOS

Característica	Descrição	Tipo
count	Conexões p/ o mesmo *host*, nos últimos 2 segundos	contínuo
	Obs: As próximas linhas referem-se as conexões de mesmo *host*	
serror_rate	% de conexões que tem erros de "SYN"	contínuo
rerror_rate	% de conexões que tem erros "REJ"	discreto
same_srv_rate	% de conexões do mesmo serviço	contínuo
diff_srv_rate	% de conexões de diferentes serviços	contínuo
srv_count	Conexões p/ o mesmo serviço, nos últimos 2 segundos	contínuo
	Obs: As próximas linhas referem-se as conexões de mesmo serviço	
srv_serror_rate	% de conexões que tem erros de "SYN"	contínuo
srv_rerror_rate	% de conexões que tem erros "REJ"	discreto
srv_diff_host_rate	% de conexões para diferentes *hosts*	contínuo

B. Pré-Processamento e Normalização

O processo de normalização e pré-processamento é, sem dúvida, uma fase muito trabalhosa e importante para garantir a eficiência de uma rede neural. A normalização dos dados foi realizada para evitar a saturação da função de ativação *Sigmoidal*. Como os resultados serão caracterizados apenas como normal ou intrusão, foram utilizados os valores 0,1 e 0,9 respectivamente, para identificá-los. Continuando o pré-processamento da base de dados, as entradas não numéricas disponibilizadas foram transformadas em entradas numéricas, para que haja a possibilidade de processamento pela rede neural. Alguns fatores tiveram de ter uma atenção especial para o tratamento dos dados, por exemplo os protocolos de serviço. Como a classificação de vários protocolos não é uma grandeza, onde é possível definir valores de distância entre eles, é possível ver no "Algoritmo 1", a implementação de um algoritmo que adiciona uma coluna diferente para cada protocolo. A coluna do protocolo do dado corrente, será preenchido com 1, enquanto as outras colunas de protocolo serão preenchidas com o valor 0. Desta forma temos *n* valores ou parâmetros de entrada de protocolos distintos encontrados na base):

- http = [1 0 0 0 0 ... *n*];
- smtp = [0 1 0 0 0 ... *n*];
- pop = [0 0 1 0 0 ... *n*];
- icmp = [0 0 0 1 0 ... *n*];

Os dados resultantes obtidos nessa etapa tinham 42 variáveis de entrada (contando o *target*), passaram a ter 118 entradas (colunas), em virtude do grande número de protocolos de serviço e outras informações.

C. Treinamento da Rede Neural

Devido as características da base de dados KDDCup'99 [26], o treinamento utilizado neste artigo foi o supervisionado. O treinamento da rede neural foi efetuado através do *Back-propagation*, já presente na biblioteca *Pybrain*. Para aprimorar a execução do algoritmo de treinamento, foi necessário definir alguns parâmetros, são eles: a arquitetura da rede, conjunto de dados de treinamento, taxa de aprendizado, taxa de momento, número máximo de épocas de treinamento, número máximo de épocas para continuar o treinamento, porção de validação (% dos dados de entrada reservados para a validação do treinamento) e função de ativação.

A função *splitWithProportion* foi utilizada para fazer a mistura dos dados e a separação para os dados de teste. Este recurso é utilizado para que o desempenho do treinamento da rede seja otimizado, pois a cada uso da função é escolhido aleatoriamente os dados de acordo com a proporção passada, variando os dados a cada iteração.

A função *trainUntilConvergence* foi utilizada para que o treinamento garanta a convergência e "escolha" o melhor período do treinamento. Isto é possível porque a função possui um parâmetro chamado *continueEpochs*, onde faz com que a função monitore durante *n* épocas (pré-definidas) se o treinamento convergiu ou não, em relação as épocas anteriores. Desta forma, é possível parar o treinamento antes de alcançar o número máximo de épocas.

Algoritmo 1: FUNÇÃO PARA PRÉ-PROCESSAMENTO DOS DADOS

Entrada: $dados\_novos, dados\_iteracao, dado$
Saída: Vetor com as colunas ajustadas

início
  $cont\_aux = len(dados\_novos)$;
  para *cada* $x \in xrange(0, len(dados\_iteracao))$ **faça**
    `// Concatena valor 0 no vetor`
    $dados\_novos.append(0)$;
  fim
  $vetor\_numerado = [..]$;
  para *cada* $protocolo \in dados_iteracao$ **faça**
    $vetor\_numerado.append([protocolo, cont\_aux])$;
    $cont\_aux \leftarrow cont\_aux + 1$;
  fim
  $cont\_aux = 0$;
  para *cada* $protocolo \in vetor\_numerado$ **faça**
    se $protocolo\,[0] == dado$ **então**
      $dados\_novos\,[vetor\_numerado\,[cont\_aux]\,[1]] \leftarrow 1$;
      $break$;
    fim
    $cont\_aux \leftarrow cont\_aux + 1$;
  fim
  retorna $dados\_novos$;
fim

Na saída da rede neural foi utilizado a função de ativação *Sigmoidal*, na qual abrange valores entre 0 e 1 (suficientes para a classificação das saídas desejadas). Devido a elevada quantidade de dados disponíveis, o número máximo de épocas definida foi 200.

O módulo *NetworkWriter* da biblioteca *Pybrain* foi utilizado para gravar a rede neural treinada, permitindo efetuar testes com diferentes bases de dados, sem precisar treinar novamente a cada novo conjunto de testes. Já para a geração do gráfico da Figura 2, houve a necessidade de alterar os retornos de algumas funções da biblioteca *Pybrain*, para que todos os dados necessários pudessem ser obtidos.

IV. RESULTADOS E DISCUSSÕES

Após o pré-processamento, construção e treinamento da rede neural pode-se perceber a alta eficiência da rede neural proposta para a solução do problema encontrado. Os erros de validação foram obtidos através dos testes da função *trainUntilConvergence* da biblioteca *Pybrain* com 20% das amostras. A utilização do parâmetro *continueEpochs* também colaborou para que bons resultados pudessem ser encontrados (Figura 2 e Tabela IV). A função percebeu que após um número de épocas pré-definidos não se obteve mais convergência e finalizou o treinamento para que não ocorra sobre-treinamento da rede (*overfitting*).

O treinamento da rede, devido a grande quantidade de dados disponibilizados pela base KDDCup'99 (mesmo utilizando apenas 10%) e pela necessidade do pré-processamento de cada entrada, teve um tempo de duração médio de 14 horas para um número máximo de 200 épocas. Os parâmetros foram escolhidos para treinamento da rede foram:

- Taxa de aprendizado: 0.01;
- Taxa de Momento: 0.9;
- Função de ativação: *Sigmoidal*;
- Máximo de épocas: 200;
- *continueEpochs*: 5;
- Número de camadas escondidas: 2;
- Quantidade de neurônios em cada camada: 10;
- Porção para validação dos dados: 10% dos dados de treinamento;
- Porção para efetuar o treinamento: 80%;
- Porção para testar o treinamento: 20%;

Como a inicialização dos pesos das entradas da rede é definida de forma aleatória, é possível contestar a taxa de acertos apresentada. Portanto, para comprovar a real eficiência da rede, o treinamento e os testes foram repetidos 10 vezes, com os mesmos parâmetros (Tabela IV). Foram utilizados 20% dos dados da base para teste (cerca de 98.000 amostras). É possível perceber que a variação da taxa de acertos é aproximada a 100% em todas as tentativas.

TABELA IV. TAXAS DE ACERTO OBTIDAS USANDO 200 ÉPOCAS

Treinamento	Taxa de Acerto	Época de Parada
1	98.87%	20
2	98.32%	32
3	98.54%	60
4	98.88%	100
5	98.57%	15
6	98.96%	53
7	98.77%	23
8	99.03%	110
9	98.71%	15
10	99.12%	96
Média	**98.77%**	**52**

TABELA V. TAXA MÉDIA DE ACERTOS APRESENTADA POR OUTROS AUTORES

Autor	Taxa Média de acertos
Naoum et. al 2012 [12]	94.7%
Sen et. al 2014 [14]	98.97%
Jing-xin et. al 2004 [13]	98.5%

Conforme apresentado na Figura 2, o erro médio quadrático no final do treinamento foi muito pequeno (próximo a zero), o que garante que, ao utilizar esta técnica em um SDIR, o número de falsos positivos e falsos negativos será bastante reduzido.

A partir da comparação das médias obtidas nas Tabelas IV e V é possível perceber que houve uma leve melhoria comparado as propostas de Jing-xin et. al 2004 [13] e Naoum et. al 2012 [12]. De qualquer forma, todas as propostas, incluindo a deste artigo, podem ser aplicadas satisfatoriamente para resolução de problemas envolvendo reconhecimento de padrões de ataques em redes de computadores.

Quanto ao resultado da classificação das amostras em termos de falsos positivos ou negativos e verdadeiros positivos ou negativos, juntamente com o número de amostras classificadas em verdadeiro e falso, são vistos logo abaixo:

- Verdadeiro Positivo: 78102;
- Verdadeiro Negativo: 19430;
- Falso Positivo: 94;
- Falso Negativo: 1179;
- Acertos: 97532;
- Erros: 1273;
- Total de amostras: 98805;

V. CONCLUSÃO

Este artigo teve como principal objetivo explorar, apresentar e validar a utilização de técnicas de redes neurais artificiais em detecção de intrusão em redes de computadores.

A biblioteca *Pybrain* supre grande parte das necessidades de implementações de redes neurais artificiais, apresentando um bom desempenho e possibilidade de melhorias através de recursos disponíveis em seu conjunto de funções. A possibilidade de edição do código (*open-source*) também é um dos motivos pelo qual, a partir de modificações na biblioteca, foi possível obter alguns valores importantes para os resultados serem atingidos.

A partir dos resultados apresentados, pode-se afirmar a viabilidade da implementação de uma rede neural, utilizando a biblioteca *Pybrain*, para resolver problemas de detecção de intrusão em redes de computadores.

Como sugestões para trabalhos futuros e incremento deste artigo pode-se alterar o algoritmo proposto para que seja possível classificar os ataques por tipo, e não somente por intrusão ou não intrusão. Fazer o pré-processamento dos dados obtidos com o *tcpdump* para poder aplicar e realizar os testes necessários em uma rede real. Propor um Sistema de Detecção de Intrusão completo, robusto, utilizando redes neurais artificiais para que seja possível sua utilização em ambientes de produção.

REFERÊNCIAS

[1] R. P. d. C. Neto, "Sistema de detecção de intrusos em ataques oriundos de botnets utilizando método de detecção híbrido," 2011.

[2] P. Barford, J. Kline, D. Plonka, and A. Ron, "A signal analysis of network traffic anomalies," in *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*, pp. 71–82, ACM, 2002.

[3] P. M. Mafra, J. da Silva Fraga, V. Moll, and A. O. Santin, "Polvo-iids: Um sistema de detecç ao de intrusao inteligente baseado em anomalias," 2008.

[4] J. Wang, *Computer network security: theory and practice*. Springer Publishing Company, Incorporated, 2009.

[5] R. P. de Azevedo, "Detecção de ataques de negação de serviço em redes de computadores através da transformada wavelet 2d," 2012.

[6] M. R. Heinen and F. S. Osório, "Autenticação de assinaturas utilizando análise de componentes principais e redes neurais artificiais," in *1st Workshop on Computational Intelligence (WCI 2006)*, pp. 1–6, 2006.

[7] E. P. de Souza and J. A. S. Monteiro, *Estudo sobre Sistema de Detecção de Intrusão por Anomalias Uma Abordagem Utilizando Redes Neurais*. PhD thesis, Master's thesis, Salvador University, Salvador, Brazil, 2008.

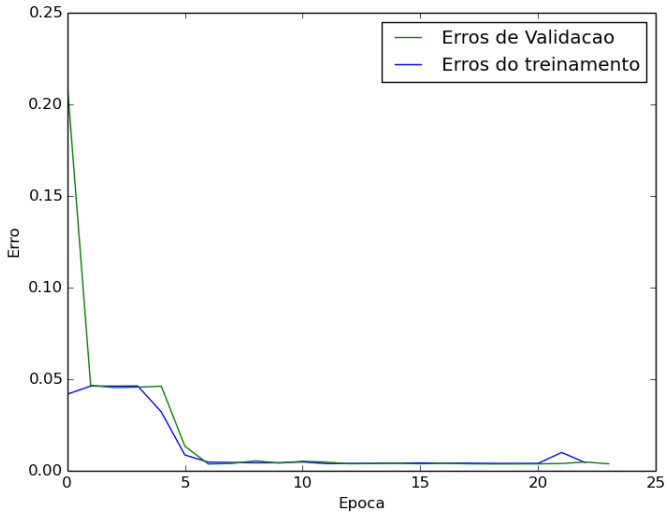

Figura 2. Gráfico representando erro médio quadrático por Época.

[8] R. G. Bace, *Intrusion detection*. Sams Publishing, 2000.

[9] Y. Yu, "A survey of anomaly intrusion detection techniques," *Journal of Computing Sciences in Colleges*, vol. 28, no. 1, pp. 9–17, 2012.

[10] K. Kafadar and E. J. Wegman, "Visualizing "typical" and "exotic" internet traffic data," *Computational statistics & data analysis*, vol. 50, no. 12, pp. 3721–3743, 2006.

[11] T. Lane, *Machine learning techniques for the domain of anomaly detection for computer security*. PhD thesis, Citeseer, 1998.

[12] R. S. Naoum, N. A. Abid, and Z. N. Al-Sultani, "An enhanced resilient backpropagation artificial neural network for intrusion detection system," *International Journal of Computer Science and Network Security*, vol. 12, no. 3, pp. 11–16, 2012.

[13] W. Jing-xin, W. Zhi-ying, and D. Kui, "A network intrusion detection system based on the artificial neural networks," in *Proceedings of the 3rd international conference on Information security*, pp. 166–170, ACM, 2004.

[14] N. Sen, R. Sen, and M. Chattopadhyay, "An effective back propagation neural network architecture for the development of an efficient anomaly based intrusion detection system," in *Computational Intelligence and Communication Networks (CICN), 2014 International Conference on*, pp. 1052–1056, Nov 2014.

[15] I. Ahmad, A. B. Abdullah, and A. S. Alghamdi, "Application of artificial neural network in detection of dos attacks," in *Proceedings of the 2nd international conference on Security of information and networks*, pp. 229–234, ACM, 2009.

[16] W. Lee and S. J. Stolfo, "A framework for constructing features and models for intrusion detection systems," *ACM transactions on Information and system security (TiSSEC)*, vol. 3, no. 4, pp. 227–261, 2000.

[17] S. S. Haykin, S. S. Haykin, and S. S. Haykin, *Kalman filtering and neural networks*. Wiley Online Library, 2001.

[18] I. V. M. d. Lima *et al.*, "Uma abordagem simplificada de detecção de intrusão baseada em redes neurais artificiais," 2005.

[19] R. Hecht-Nielsen, "Theory of the backpropagation neural network," in *Neural Networks, 1989. IJCNN., International Joint Conference on*, pp. 593–605, IEEE, 1989.

[20] A. de Pádua Braga, A. C. P. de Leon Ferreira, and T. B. Ludermir, *Redes neurais artificiais: teoria e aplicações*. LTC Editora, 2007.

[21] C. M. Bishop, *Neural networks for pattern recognition*. Oxford university press, 1995.

[22] F. Sehnke, C. Osendorfer, T. Rückstieß, A. Graves, J. Peters, and J. Schmidhuber, "Policy gradients with parameter-based exploration for control," in *Artificial Neural Networks-ICANN 2008*, pp. 387–396, Springer, 2008.

[23] T. Rückstieß, M. Felder, and J. Schmidhuber, "State-dependent exploration for policy gradient methods," in *Machine Learning and Knowledge Discovery in Databases*, pp. 234–249, Springer, 2008.

[24] T. Schaul and J. Schmidhuber, "Scalable neural networks for board games," in *Artificial Neural Networks–ICANN 2009*, pp. 1005–1014, Springer, 2009.

[25] T. Schaul, J. Bayer, D. Wierstra, Y. Sun, M. Felder, F. Sehnke, T. Rückstieß, and J. Schmidhuber, "Pybrain," *The Journal of Machine Learning Research*, vol. 11, pp. 743–746, 2010.

[26] M. Tavallaee, E. Bagheri, W. Lu, and A.-A. Ghorbani, "A detailed analysis of the kdd cup 99 data set," in *Proceedings of the Second IEEE Symposium on Computational Intelligence for Security and Defence Applications 2009*, 2009.

[27] M. Sabhnani and G. Serpen, "Kdd feature set complaint heuristic rules for r2l attack detection.," in *Security and Management*, pp. 310–316, Citeseer, 2003.

[28] Zhao Yanling and Deng Bimin and Wang Zhanrong, Analysis and study of perceptron to solve XOR problem in *Autonomous Decentralized System,* 2002. *The 2nd International Workshop on*

[29] The 3rd International Knowledge Discovery and Data Mining Tools Competition, *http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html*, 2003