Universidade Federal de Itajubá
UNIFEI

PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

Detecção de Intrusão Utilizando Redes Neurais Artificiais no Reconhecimento de Padrões de Ataque

ROBERTO SILVA NETTO

Itajubá, dezembro de 2006

Universidade Federal de Itajubá
UNIFEI

ROBERTO SILVA NETTO

Detecção de Intrusão Utilizando Redes Neurais Artificiais no Reconhecimento de Padrões de Ataque

Dissertação apresentada à Universidade Federal de Itajubá para obtenção do título de Mestre em Engenharia Elétrica.

Área de Concentração:
Automação e Sistemas Elétricos Industriais
Orientador: Prof. Dr. Otavio Augusto Salgado Carpinteiro

Itajubá 2006

Dedico este trabalho a Deus, aos meus pais pelo amor e a minha esposa, carinho e dedicação presentes em cada um dos seus atos e conselhos.

Agradecimentos

Agradeço a todos que de alguma forma contribuíram para o desenvolvimento deste trabalho, em especial:

A Deus por estar sempre presente na minha vida, me ajudando e capacitando, que me deu forças quando precisei, que me consolou na minha tristeza e que sempre tem me dado alegrias.

Aos meus pais, Lineu e Anna, por terem me educado e me darem apoio em todos os sentidos, por acreditarem no meu sonho e serem responsáveis pela minha formação.

A minha amada esposa Fernanda, companheira, amiga, com a qual dividi e compartilhei as minhas tristezas, alegrias e que também é responsável por mais esta vitória.

Aos meus sogros Eron e Edna, que também foram meus pais e em todos os momentos em que estive com eles, me trataram como filho.

Aos meus sócios Caio, Cezar e Rodrigo, por dividirem seus sonhos comigo para realizar algo maior.

Aos amados mestres Maurílio e Otavio por acreditarem na minha capacidade e me instruírem com algo a mais que levarei comigo no coração e aos professores Edmilson e Isaías que ajudaram na conclusão do trabalho.

E ao meu filho, que me fez enxergar a vida com outros olhos.

Resumo

O número de ataques cibernéticos vem crescendo cada vez mais, pois as ferramentas de invasão estão cada vez mais fáceis de encontrar e as vulnerabilidades encontradas, demoram a serem corrigidas. Em virtude disso, os sistemas de detecção de intrusão têm-se tornado um componente necessário na maioria dos sistemas de segurança de rede. O principal objetivo é identificar potenciais violações nas políticas de segurança. A maioria dos sistemas de detecção de intrusão é baseada em regras e necessitam que a base de dados seja atualizada a cada vez que um novo ataque é descoberto. Entretanto esta tarefa não é simples, pois as regras são complicadas e necessitam de conhecimento profundo do ataque que se deseja detectar, além de consumir um tempo importante do administrador do sistema que deverá fazer a verificação da base de dados constantemente. Caso o sistema de detecção de intrusão não seja atualizado, este se torna uma brecha no sistema de segurança, pois não irá relatar uma invasão quando a rede for invadida por um ataque desconhecido. Este trabalho tem como propósito apresentar o uso de redes neurais artificiais no problema de detecção de tais violações e em novos padrões de ataque. Para isso foi utilizado uma rede Multi-Layer Perceptron (MLP) com o intuito de reconhecer padrões de ataques de rede, tendo como base de dados o Third International Knowledge Discovery and Data Mining Tools Competition. Como essa base de dados possui campos discretos e contínuos, foi necessária a normalização dos 41 campos que a compõe para viabilizar o seu uso no treinamento das redes neurais. Optou-se pela utilização de 4 redes neurais, as quais tinham como objetivo detectar um tipo de ataque, Remote-to-Local, User-to-Root, Probe e DoS, além de reconhecer corretamente os padrões de tráfego normal. Obtiveram-se ótimos resultados na utilização de redes neurais no reconhecimento de padrões de ataque, com altas taxa de detecção dos novos ataques e baixas taxas de falsos positivos e falsos negativos, tendo um caso onde ocorre 100% de detecção de ataque e 100% de detecção de tráfego normal.

Abstract

The number of cyber attack is growing up more and more, because, the tools of invasion is become more widely e easy to find and the vulnerabilities take a long time to be corrected. For all this, the Intrusion Detection System has become a necessary device in the most of network security system. The main objective is identifying potential violations in security policy. The most Intrusion Detection System are based on rules and need that tha database be update every time that a new attack is discovered. However, this task is not too simple, since the rules are complicated and require deep knowledge of the attack that we wish detect, further than will consume important time of the system administrator, who should make the database update frequently. If the intrusion detection system not be updated, this system become a security flaw, because it will not report a invasion when the network be invaded by a unknown attack. This work has the purpose of show the use of artificial neural network for the problem of detection of those violations and newer attack pattern. For this, was used a Multi_layer Perceptorn (MLP) network whit the intuit of recognize network attack patterns, having whit database the Third International Knowledge Discover and Data Mining Tools Competition. As this database possesss discrete and continuous fields, the normalization of the 41 fields was necessary that composes it to make possible its use in the training of the neural nets. It was opted to the use of 4 neural nets, which had as objective to detect a type of attack, Remote-to- Local, User-to - the Root, Probe and Of, beyond recognizing the standards of normal traffic correctly. Excellent results in the use of neural nets in the recognition of standards of attack had been gotten, with high rate of detection of the new attacks and decreases taxes of false positives and false negatives, having a case where 100% of attack detention occur and 100% of detention of normal traffic.

Lista de Abreviaturas e Siglas

IDS – INTRUSION DETECTION SYSTEM

MLP – MULTI LAYER PERCEPTRON

KDD – KNOWLEDGE DISCOVERY AND DATA MINING

DARPA – DEFENSE ADVANCED RESEARCH PROJECTS AGENCY

NIDS – NETWORK BASED INTRUSION DETECTION SYSTEM

HIDS – HOST BASED INTRUSION DETECTION SYSTEM

LAN – LOCAL AREA NETWORK

SOM – SELF ORGANIZED MAPS

MARS - MULTIVARIATE ADAPTATIVE REGRESSION SPLINES

SVM – SUPORT VECTOR MACHINE

RN – REDES NEURAIS

RT – REGRESSION TREE

DOS – DENIAL OF SERVICE

R2L – REMOTE TO LOCAL

U2R – USER TO ROOT

Sumário

Lista de Figuras

Lista de Tabelas

1 INTRODUÇÃO

Os sistemas de detecção de intrusão têm-se tornado um componente necessário na maioria dos sistemas de segurança de rede. O seu principal objetivo é identificar potenciais violações nas políticas de segurança [1]. A primeira proposta de um IDS foi feita por Dorothy E. Denning[2] em 1986, apresentando um modelo para a detecção de tentativas de invasão, penetração e outros tipos de abusos no uso do computador. Desde então, têm aparecido várias propostas para suprir as necessidades e deficiências dos IDS, como o NetRanger [3] e NID [4] e, posteriormente, o BRO [5] e NetStat [6], ambos baseados na verificação de assinaturas. Porém, estes sistemas necessitam de atualização sempre que um novo ataque é descoberto, o que dificulta o trabalho do administrador da rede; pois além de observar os relatórios dos IDS, precisa estar atento para as atualizações de banco de dados das assinaturas de ataques.

1.1 Considerações Iniciais

Existem vários tipos de IDS´s e também várias maneiras em que eles podem ser inseridos em uma rede de computadores. Este trabalho é baseado no NIDS, ou simplesmente *Network Intrusion Detection System*, que tem como objetivo detectar

intrusões através de informações coletadas a partir da rede em que o equipamento se encontra conectado.

Um problema encontrado reside no fato dos IDS´s disponíveis no mercado, como o Snort, serem na maioria baseados em regras (*ruled-based*), e com isso eles não são capazes de satisfazer exigências como precisão, eficiência e adaptabilidade automática simultaneamente [2]. Sendo assim, eles apenas reconhecem os ataques descritos em sua base de dados e qualquer mudança ou alteração das características dos ataques, fará com que a intrusão passe despercebida pelo IDS [6]. Pode ocorrer também que o invasor use um método desconhecido da base de dados do IDS, isso pode ocorrer devido a não atualização, ou para que os desenvolvedores do IDS criassem novas regras a serem inseridas no banco de dados do IDS. Desse modo, tem-se um falso positivo, ou seja, um ataque sendo classificado como um tráfego normal.

Como a finalidade básica de um IDS é detectar um ataque que está para acontecer, ao ser enganado pelo atacante ou pela utilização de alguma ferramenta desconhecida que explora as falhas, também conhecida como *exploit*, o IDS passa a ser uma brecha no sistema de segurança de uma rede. Essa brecha ocorre, pois, os administradores ao monitorarem a rede através de respostas do IDS, e não há nenhuma adversidade encontrada, o invasor pode completar sua missão totalmente coberto pelo IDS, já que este não executou seu papel.

Em problemas como esses, o trabalho propõe o uso de redes neurais na detecção de intrusão, para que os novos ataques possam ser detectados, sem a necessidade de atualização rotineiramente da base de dados. Isto é possível pela característica das redes neurais em reconhecer padrões. Uma rede MLP (*Multi Layer Perceptron*) foi usada como proposta e o conjunto de teste, a base do KDD Cup99, que é uma sub-versão do DARPA98, utilizada em vários experimentos para avaliação de desempenho de modelos de IDS. Como os ataques são classificados em quatro categorias, optou-se por uma rede para cada tipo de ataque.

No capítulo 2 é apresentado o funcionamento do IDS, quais os modos em que ele pode ser configurado dentro de uma empresa para a detecção de intrusão e quais os mecanismos que ele utiliza para a obtenção da informação que será utilizada para a detecção. Também é apresentado como ele realiza a classificação

por *anomaly,* buscando reconhecer desvios do padrão normal, ou *misuse*, que busca a identificação de mal uso de recursos.

No capítulo 3 apresentam-se os conceitos básicos de redes neurais artificiais, o modelo de neurônio artificial, o modelo e os processos de aprendizagem de uma rede neural e como é realizada a aprendizagem. Além disso, o *perceptron* e a arquitetura de rede do *Multi Layer Perceptron*, que é usada no trabalho, bem como os algoritmos de retropropagação, de treinamento, a função de ativação, validação cruzada e como ocorre a generalização.

No capítulo 4 relatam-se os experimentos e os resultados, a base de dados utilizada, os *exploits e* quais foram usados na base de dados, como foi feita a taxonomia dos *exploits,* o tratamento da base, os dados para treinamento da rede e, finalmente, os resultados obtidos com a utilização das redes neurais na detecção de intrusão.

O último capítulo apresenta a conclusão do trabalho, baseado nos resultados e na proposta apresentados e possíveis futuros trabalhos.

2 IDS SISTEMA DE DETECÇÃO DE INTRUSÃO

2.1 Sistemas de Detecção de Intrusos

Uma intrusão é um acesso ou uso não autorizado de um recurso em um sistema computacional. IDS são *softwares* que tem como função detectar, identificar e responder a atividades anormais e não autorizadas em um sistema. O objetivo do IDS é prover um mecanismo para detecção de violações de segurança tanto em tempo real, como em *batch-mode*. As violações podem ser iniciadas tanto por pessoas de fora da rede, tentando entrar no sistema, quanto por pessoas internas, tentando fazer mau uso de seus privilégios. IDS coleta informações de uma variedade de sistemas e fontes de rede, e então analisa as informações, procurando sinais de intrusão e mau uso. As principais funções de um IDS são:

- Monitorar e analisar atividades do usuário e do sistema;
- Avaliar a integridade de sistemas críticos e dados de arquivos;
- Reconhecer padrões de atividades que refletem ataques conhecidos;
- Responder automaticamente à atividade detectada; e
- Reportar o resultado do processo de detecção.

Detecção de intrusos pode ser dividida em duas amplas categorias, baseadas no método de detecção: *misuse detection* e *anomaly detection*. *Misuse detection* funciona com a procura de traços ou padrões de ataques conhecidos. Claramente, apenas ataques conhecidos, que deixam traços característicos podem ser detectados neste modo. *Anomaly detection*, por outro lado, utiliza um modelo de comportamento normal do usuário ou do sistema, e acusa desvios significativos deste modelo como potencialmente malicioso. Este modelo de comportamento normal do usuário ou sistema é comumente chamado de perfil de usuário ou sistema. A maior vantagem do *anomaly detection* é a habilidade de detectar ataques previamente desconhecido.

IDS são categorizados de acordo com a localização da fonte de auditoria que eles analisam. A maioria dos IDS são classificados como detecção de intrusão baseado em *host (host-based)* ou detecção de intrusão baseado em rede (*network-based)* para reconhecimento e tratamento de ataques [7].

Um IDS é um sistema de monitoramento que pode estar sendo executado tanto na máquina alvo para análise de seus padrões de tráfego, ou em uma máquina independente, para análise e monitoração de padrões de tráfego de rede. Devido a falhas de softwares, sistemas mal configurados, ferramentas de quebra de senhas e outras falhas de projeto, alguns intrusos podem invadir computadores facilmente. Assim, o desenvolvimento de um sistema de detecção de intrusão confiável para garantir a segurança de uma rede de computadores é crítico.

Isto é difícil para os tradicionais IDS´s, como os baseados em regras ou em estatísticas, que exigem alta precisão e adaptabilidade automática ao mesmo tempo. Um IDS baseado em regras pode detectar alguns ataques conhecidos com uma alta taxa de detecção, mas é difícil detectar novos ataques, e a sua base de dados de assinaturas precisa ser atualizada freqüentemente. Um IDS baseado em estatística deve coletar dados suficientes para compor um complicado modelo matemático, que é difícil e algumas vezes impraticável no caso de tráfego de redes complexo. As Redes Neurais Artificiais provêem um novo método de projeto para o IDS. As RNA´s são especialmente úteis quando o conhecimento explícito necessário para construir um IDS não está disponível. IDS baseados em redes neurais podem ser usados tanto para detecção em *misuse* quanto em *anomaly*. Usualmente, o *misuse detection* possui uma taxa maior de detecção que o *anomaly detection*, porém ele

não é capaz de detectar ataques novos antes que sua estrutura seja atualizada. Por outro lado, o *anomaly detection* pode classificar a maioria dos registros anormais, contudo, ele não pode identificar qual o tipo específico de intrusão é o registro.[8].

2.2 O que é uma intrusão?

Para entender o que é “detecção de intrusão”, faz-se necessário, primeiramente, entender o que é uma intrusão. O dicionário Houasis define uma intrusão como “o ato de empurrar, ou entrar em um lugar ou estado sem ser convidado, sem ter direito ou ser bem vindo.” Para nossos propósitos, é simplesmente uma atividade não autorizada no sistema ou na rede por um de seus computadores ou redes de computadores. Isto pode incluir um usuário legítimo do sistema, tentando escalonar seus privilégios e ganhar maior acesso do que foi permitido a um usuário remoto e não autenticado, comprometendo um serviço em execução, a fim de criar uma conta no sistema. Uma intrusão também pode ser um vírus proliferando através do seu sistema de e-mail e outros cenários similares. As intrusões podem vir de qualquer invasor deliberadamente malicioso, ou de um usuário final qualquer. Por exemplo, ao se abrir anexos de e-mails enviados também, apesar de repetidas advertências para não fazerem. As intrusões também podem vir de alguém totalmente desconhecido, em outro continente, de um ex-empregado, ou do próprio pessoal de confiança [9].

2.3 “Scanning versus Compromise”

Quando se observa uma rede de computadores, uma das primeiras atividades que sobressai é a de *scanning*, que consiste em vascular uma rede ou um computador à procura de vulnerabilidades. O *scanning* por vulnerabilidades, particulares ou apenas por portas abertas, é atividade comum na Internet sem filtro e em muitas redes privadas. Muitos IDS são configurados para indicar atividade de *scanning*, e não é incomum receber alarmes causados por alguma forma de *scanning*. Apesar do *scanning* não ser necessariamente uma atividade maliciosa, pois pode ter causas legítimas (um administrador local checando sua própria rede

por vulnerabilidades para aplicação de correção; por exemplo, uma empresa terceirizada contratada para executar uma auditoria de segurança no sistema), é uma técnica freqüentemente utilizada como prelúdio de uma tentativa de ataque. Sendo assim, muitos administradores desejam ser alertados quando o sistema estiver sendo vasculhado. Rastrear atividade de *scanning* também pode ser útil para correlação em futuros ataques. Entretanto, é importante destacar que o *scanning* não é uma tentativa comprometedora por si só, e não deve ser tratada com o mesmo nível de resposta que uma tentativa real de ataque. É importante notar também que a atividade de *scanning* esta se tornando constante, e atualmente encontram-se programas automatizados para vasculhar grandes faixas de endereços. Ferramentas de monitoramento de redes, *worms* e vírus, aplicações de otimização automática, *script kiddies*, e muito mais estão constantemente verificando máquinas na rede [9].

Nesse contexto, faz-se necessário um esforço conjugado na filtragem e análise do tipo tráfego na Internet.

2.3.1 Virus e Worms—SQL Slammer

Outro tipo muito comum de tráfego que será visto ativando o IDS é o worm. Worms e vírus são freqüentemente bons candidatos para o IDS, pois eles possuem comportamento repetitivo e identificável. Mesmo os vermes polimorfos e os vírus que utilizam muitos vetores de ataque terão algum comportamento de rede em comum, algum modelo de tráfego que pode ser combinado e descoberto pelo seu IDS. Como um exemplo, tem-se o worm SQL Slammer. No dia 25 de Janeiro de 2003, o SQL slammer foi lançado na Internet, também conhecido como *Sapphire* [ref](Safira), o verme explora uma fraqueza no servidor *Microsoft Structured Query Language* (SQL). Ele envia um pacote *User Datagram Protocol* (UDP) de 376 bytes para a porta 1434, inunda um buffer no servidor SQL, e ganha privilégios de Sistema, o nível mais alto possível do usuário em um sistema operacional Windows. Uma vez que ele comprometeu com sucesso um *host*, ele começa a vasculhar outros endereços IP para estender-se além disso.

A partir do momento do seu lançamento, o SQL *Slammer* alcançou toda a rede mundial em aproximadamente 10 minutos. Nesse intervalo de tempo,

montantes maciços de largura de banda da rede foram consumidos pelas tentativas de exploração e propagação do verme. Muitos sistemas foram comprometidos. Cinco dos 13 principais servidores de Nome de Domínio, que fornecem o serviço à Internet, foram derrubados.

Dentro desse contexto, tem-se o desafio de identificar o padrão de ataque para este tipo de comportamento. Obviamente, isto é o tipo de atividade que deseja-se descobrir na rede. Uma característica comum entre cada *host* infectado pelo *Slammer* é o *payload* com *exploit* que ele distribui. E de fato, é exatamente esta assinatura que a regra do IDS Snort quer pegar. A seguir, figura 2-4 está a assinatura do Snort que combina com esta atividade:

```
alert udp $EXTERNAL_NET any -> $HOME_NET 1434 (msg:"MS-SQL Worm propagation attempt"; content:"|04|"; depth:1; content:"|81 F1 03 01 04 9B 81 F1 01|";content:"sock"; content:"send"; reference:bugtraq,5310; classtype:miscattack; reference:bugtraq,5311; reference :url,vil. /vil/content nai.com /v_99992.htm; sid:2003; rev:2;)
```

Figura 2-1 Regra do Snort para o SQL Slammer

Usando esta assinatura, pode-se descobrir e enumerar quantas tentativas de ataque são detectadas, e quais *hosts* eles tentavam alcançar. Os ataques maciços automatizados como este, normalmente engendra uma resposta coordenada da comunidade de segurança - programadores de IDS que escrevem novas assinaturas, vendedores de antivírus escrevendo atualizações e correções, provedores de *backbone* que seguem a pista do tráfego e mitigam o seu efeito, filtrando quando solicitado e quanto necessário. Esta assinatura pode ajudar a seguir a pista de tentativas de infecção pelo verme na rede, e assegurar que os sistemas sob ataque permaneçam seguros. A coordenação de respostas entre companhias e defensores é um dos poucos modos que se podem acompanhar os atacantes. Um grande número de organizações se dedicam a ajudar os respondentes a lidar com ataques e compartilhar informação [9].

2.3.2 Live Attacks—Sendmail Buffer Overflow

Na seção anterior, viu-se como detectar um ataque através da utilização de regras para detectar o *exploit* dentro do *payload*. Entretanto existem outras formas de ataques como, por exemplo, uma tentativa no transbordamento de um serviço do Wingate POP3.

A vulnerabilidade explorável desse aplicativo é a possibilidade de ultrapassar os limites de buffer remotamente na implementação do *daemon* Wingate POP3. Depois que um comando “USER” é enviado, uma quantidade suficientemente grande de dados irá sobrepor o *buffer,* podendo levar à execução independentemente de um código exploit que foi inserido. O uso normal do *daemon* POP3 forneceria somente um *username* depois da ordem de USER, e um username normal, improvavelmente, seria muito longo. Na figura 2.2 pode-se ver a regra do Snort que descobre esta tentativa de *exploit*:

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 110 (msg:"POP3 USER overflow attempt";
flow:to_server,established; content:"USER"; nocase; isdataat:50,relative;
pcre:"/^USER\s[^\n]{50,}/smi"; reference:bugtraq,789; reference:cve,CVE-1999-0494;
reference:nessus,10311; classtype:attemptedadmin; sid:1866; rev:7;)
```

Figura 2-2 Regra do Snort para Wingate POP3 buffer overfow

Esta regra procura dados contendo USER seguido por mais de 50 bytes de dados, onde aqueles 50 bytes de dados depois de USER não contêm um caractere de nova linha. Isto deve combinar com o modelo de dados que se vê em uma tentativa verdadeira de estouro do *buffer*, e não deve combinar com senhas de entrada de usuários legítimos.

2.4 Como funciona um IDS

Como já foi visto algumas das capacidades de um IDS, de alertar sobre tráfego malicioso, nesta seção será mostrado qual fonte de dados um IDS deve usar para monitorar e separar tráfego normal de ataque, e algumas possíveis respostas quando é percebido tráfego malicioso.

O IDS depende enormemente de seu tipo do local em que ele é colocado na rede. IDS´s são classificados pelas suas funcionalidades e são agrupados em 3 categorias:

- Sistema de Detecção de Intrusão Baseado em Rede (NIDS)
- Sistema de Detecção de Intrusão Baseado em Host (HIDS)
- Sistema de Detecção de Intrusão Distribuído (DIDS)

2.4.1 IDS de Rede (NIDS)

O nome NIDS vem do fato do monitoramento de um segmento inteiro de uma rede ou uma sub-rede. Isto é feito alterando o modo da placa de rede(NIC) do NIDS. Normalmente um NIC opera em modo não promíscuo, escutando apenas os pacotes destinados para o seu próprio endereço de controle de acesso ao meio ou *MAC address*. Os outros pacotes não são encaminhados para as camadas superiores da pilha para análise, eles são ignorados. Para monitoramento de todo o tráfego da sub-rede, não apenas aqueles pacotes endereçados à máquina do NIDS, ele deve aceitar todos os pacotes e encaminhá-los às camadas superiores da pilha - isto é conhecido como modo promíscuo.

No modo promíscuo, o NIDS pode escutar, sem ser percebido, toda comunicação do segmento de rede. Entretanto, isto não é tudo que é necessário para garantir que o NIDS seja capaz de escutar todo o tráfego da subrede. O equipamento de rede imediatamente acima do NIDS também deve ser configurado para enviar todos os pacotes da subrede para o NIDS. Se o equipamento é um *hub*, todos os pacotes são automaticamente enviados, pois todas as portas do *hub* recebem todo tráfego por broadcast. Entretanto, se o equipamento é uma switch, deve-se colocar a porta que está ligada à *switch* no modo de monitoramento, tornando-a uma porta de *span*. Após configurar o NIDS, é aconselhável executar uma ferramenta *sniffer* na interface, para garantir que se possa ver todo o tráfego da subrede.

A vantagem de um NIDS é que ele não impacta o sistema ou a rede pelo monitoramento. Ele não adiciona nenhuma carga aos *host*, e um atacante que comprometa um dos sistemas observados não pode tocar no NIDS, e até não sabe

de sua presença. Um lado negativo do monitoramento é usar o máximo da porta de span que está designada a uma rede, e utilizar o máximo da largura de banda na porta do NIDS. Quando se tem 20(vinte) portas de 100MB jogando para uma única porta, o *backplane* ficará saturado e se estiver além de seu limite haverá perda pacotes.

Devido a regulamentos de privacidade emergentes, o monitoramento de comunicações de rede é uma responsabilidade que deve ser ponderada cuidadosamente. Deve-se ter certeza de que o administrador está familiarizado com as exigências legais e locais para tal atividade.

Na figura 2-3, é apresentada uma rede usando três NIDS. As unidades foram colocadas em segmentos de rede estratégicos e podem monitorar o tráfego de rede de todos os equipamentos no segmento. Esta configuração representa uma topologia de rede padrão de perímetro de segurança, onde as sub-redes filtradas dos servidores públicos são protegidas pelos NIDS. Quando um servidor público é comprometido numa sub-rede filtrada, o servidor pode se tornar uma plataforma de lançamentos para adicionais *exploits*. É necessário um monitoramento cuidadoso para prevenir danos futuros.

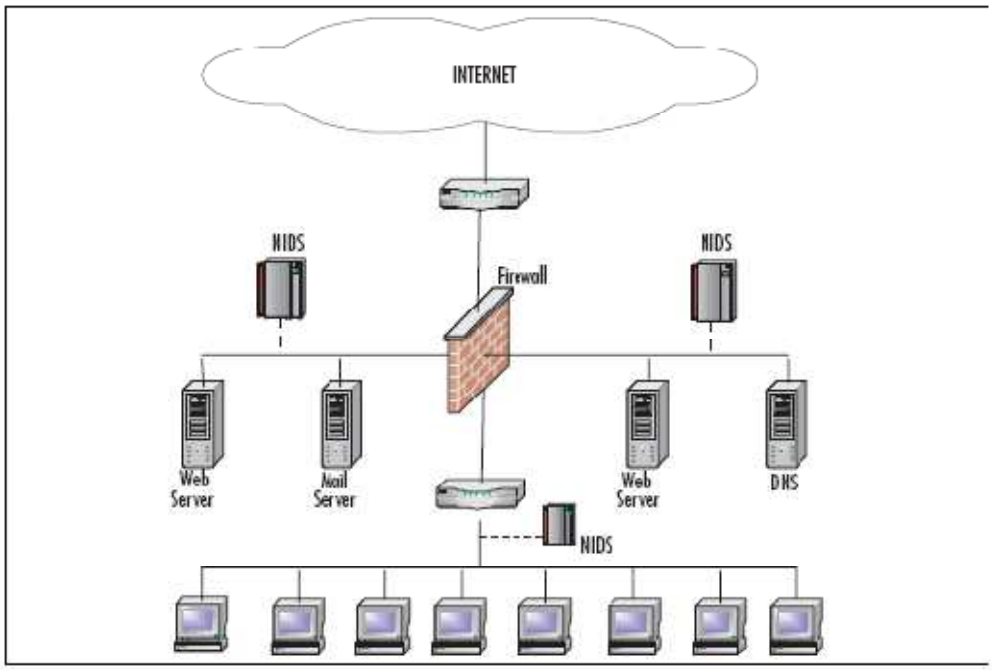

Figura 2-3 Rede com o NIDS

2.4.2 IDS baseado em Host (Host-Based)

Host based IDSs, ou HIDSs, diferenciam-se de NIDSs de dois modos: primeiro, um HIDS instalado protege só o sistema no qual ele reside, não a subrede inteira; segundo, o NIC de um sistema com um HIDS instalado, funciona normalmente no modo não-promíscuo. Isto pode ser uma vantagem em alguns casos, pois nem todos os NICs podem operar no modo promíscuo, embora NICs mais modernos possam. Além do mais, o modo promíscuo pode fazer uso direto da CPU e tornar a máquina lenta.

Outra vantagem do HIDS é a capacidade de refinar o conjunto de regras (*ruleset)* e ser bem específica para um determinado sistema do *host*. Por exemplo, não há nenhuma necessidade em configurar múltiplas regras projetadas para descobrir os *exploit* do Sistema de Arquivo de Rede (NFS) em um *host* que não esteja usando o NFS. Ser capaz de efetuar o controle detalhado do ruleset, aumenta o desempenho e reduz os falsos positivos (ou positivos verdadeiros que simplesmente não preocupa). A principal vantagem de um HIDS, contudo, reside na sua capacidade de descobrir modificações específicas nos arquivos e no sistema

operacional do *host*. Ele pode controlar tamanhos de arquivos e *checksums* para assegurar que os arquivos de sistemas cruciais não sejam maliciosamente modificados sem ninguém notar. Pode também interceptar chamadas do sistema, que podem ser uma tentativa na exploração de uma vulnerabilidade local. Além disso, ele pode monitorar o tráfego dentro de um sistema que nunca cruza a rede, e por isso, nunca seria visto pelo NIDS.

Há alguns aspectos negativos em se usar um HIDS. Deve-se escolher aquele que é configurado ao sistema operacional em uso. Um HIDS acrescentará uma carga adicional ao host no qual é configurado, isto normalmente não é um problema no *desktop* de um indivíduo, mas pode se tornar em um servidor de rede. Deve-se assegurar certa familiaridade aos detalhes de um HIDS, com a escolha e como funciona – alguns HIDS verificarão os acessos a arquivos, tempos de uso, cargas de processos, e/ou chamadas do sistema; enquanto outros também podem verificar a atividade de rede a partir daquele *host*. Garantir quais características desejadas no HIDS, e assegurar que o HIDS selecionado suportará tais características em todas as plataformas em que se necessite.

Além disso, a manutenção de uma grande rede com muitos HIDS instalados pode ser um desafio. A solução HIDS sozinha nem sempre funciona bem, e sem uma gerência centralizada pode tornar o administrador de sistema ocupado demais na tentativa de acompanhar os alertas.

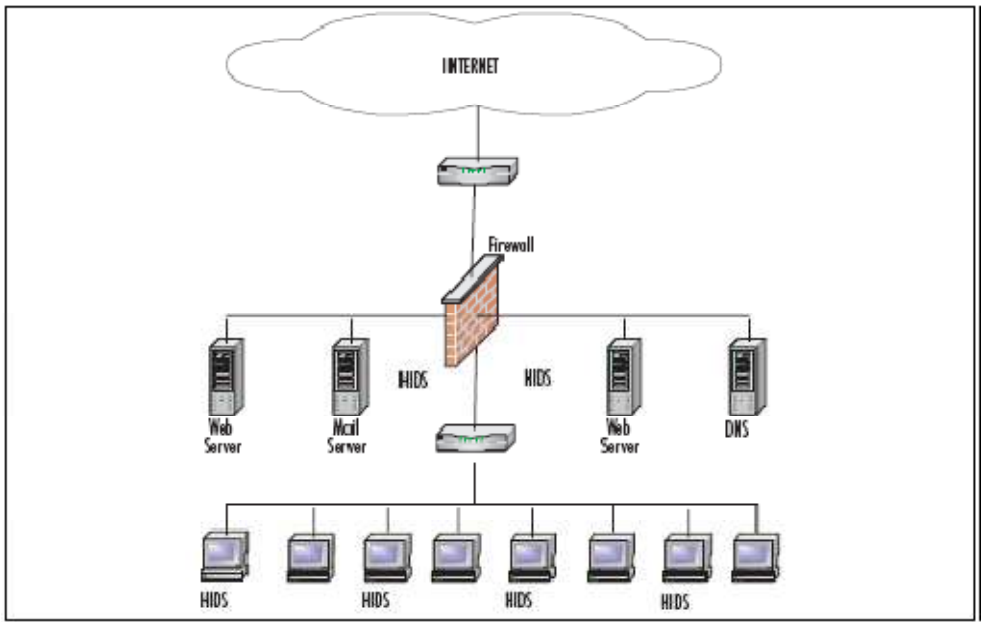

Figura 2-4 Rede com HIDS

Como citado anteriormente, o *ruleset* do HIDS no servidor de correio é construído para protegê-lo de *exploits* do servidor de correio, enquanto as regras de servidor da Web são destinadas para *exploits* da Web. Durante a instalação, as máquinas individuais podem ser configuradas com um conjunto comum de regras, e as novas regras podem ser carregadas periodicamente para prestar contas de novas vulnerabilidades.

2.4.3 Distributed IDS

Um Sistema de Detecção de Intrusão Distribuído, ou DIDS, é uma combinação de sensores NIDS, sensores de HIDS, ou ambos, distribuídos através de uma empresa e reportada a um sistema de correlação central. Os *logs* de ataques são gerados nos sensores e transferidos (periodicamente ou continuamente) à estação de servidor central, onde eles podem ser armazenados em um banco de dados central. As novas assinaturas de ataques são criadas ou carregadas na estação de gerência, assim que elas ficam disponíveis e, então, podem ser novamente carregados nos sensores de acordo com as necessidades. As espécies diferentes de sensores podem ou não ser gerenciadas pelo mesmo servidor, e os servidores de gerência são freqüentemente separados dos servidores

que reúnem os *logs*. As regras de cada sensor podem ser destinadas para encontrar as suas necessidades individuais, ajustando-se à rede ou ao host que cada sensor controla. Os alertas podem ser encaminhados a um sistema de mensagem localizado na estação do sistema de correlação e usados para notificar o administrador do IDS.

Na Figura 2.5, vê-se um sistema DIDS contendo quatro sensores e uma estação de gerência centralizada. Sensores NIDS 1 e NIDS 2 estão funcionando no modo promíscuo e estão protegendo os servidores públicos. Os Sensores NIDS 3 e NIDS 4 estão protegendo os sistemas da base computacional confinada.

As transações de rede entre sensor e gerente podem estar em uma rede privada, como representado, ou o tráfego de rede pode usar a infra-estrutura existente. Usando a rede existente para dados de gerência, o uso de segurança adicional através de criptografia, ou tecnologia VPN, é altamente recomendada. Enviar toda a informação sobre segurança da sua rede, através dela em texto ASCII ou *cleartext*, é sugerir ao invasor para interceptar tais comunicações. Na melhor das hipóteses, elas podem entregar, onde está colocado o IDS, e fornecer o comportamento para evitar a detecção. Na pior das hipóteses, podem interceptar e modificar o mecanismo de alerta, corrompendo os dados e a possibilidade de confiabilidade de análise e/ou processamento.

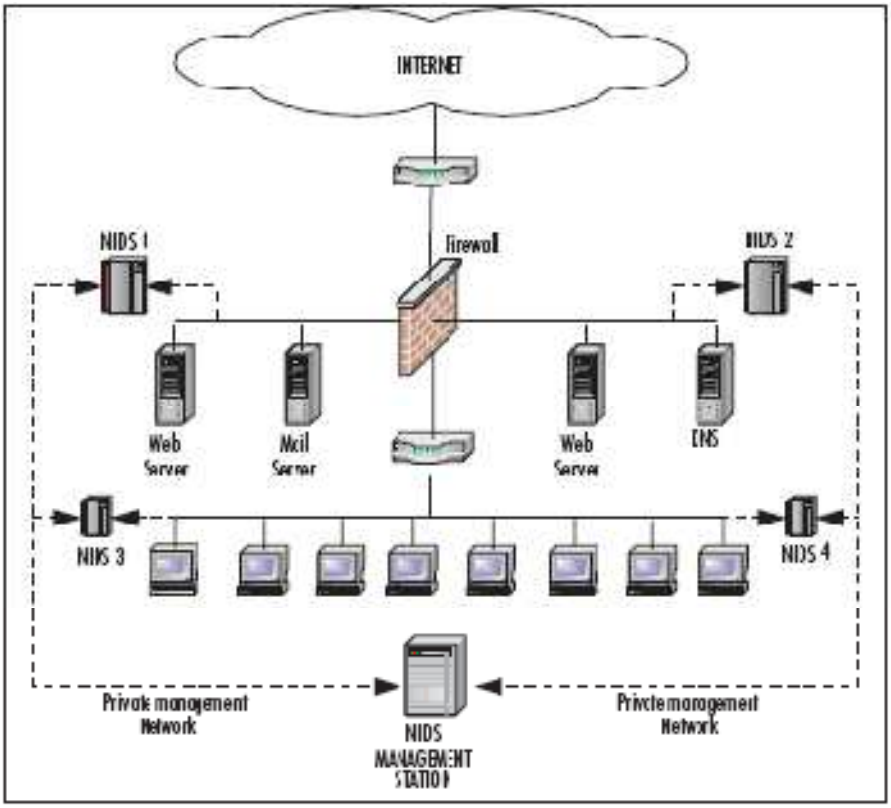

Figura 2-5 Rede com DIDS

Outra questão para se ter em mente é fazer com que o DIDS comuniquem-se através da rede comum, pois a rede pode ser inundada ou inutilizada pelo tráfego malicioso (como aconteceu em muitas redes em conseqüência do SQL Slammer). Deste modo os seus sensores IDS não serão capazes de comunicar-se com a correlação central ou com os servidores de gerência, significativamente reduzindo a sua utilidade.

Uma das principais vantagens de analisar eventos que usam DIDS é a observância de incidentes em todo o sistema, ou até em toda a Internet.

O DIDSs podem ser complexos no projeto, e necessitam de habilidades nos ajustes de correlação e gerenciamento dos dados que são gerados em todos os sensores. O alcance e a funcionalidade do sistema variam muito de implementação para implementação, e os sensores individuais podem ser NIDS, HIDS, ou uma combinação deles dois. O sensor pode funcionar em modo promíscuo ou modo não-promíscuo.

2.4.4 Informação de uma Aplicação Específica

Os três tipos de IDSs são capazes de monitorar pelo menos alguma informação específica de uma aplicação. Isto pode variar do tráfego que vai do servidor da Web às estruturas de dados internas da aplicação proprietária. Naturalmente, para uma aplicação personalizada, deve-se ter regras personalizadas no IDS, para combinar com o tráfego. Como o tráfego da aplicação vai pelo cabo através da rede, o NIDS será capaz de descobri-lo. Se houve envio em *cleartext* como tráfego de Telnet ou HiperText Protocol (HTTP), o NIDS não deve ter nenhum problema na detecção. Por exemplo, na figura 2-6 tem-se uma assinatura na procura de acesso a um PHP vulnerável.

```
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"WEB-PHP
Advanced Poll admin_tpl_misc_new.php access"; flow:to_server,established;
uricontent:"/admin_tpl_misc_new.php"; nocase; reference:bugtraq,8890;
classtype:web-application-activity; sid:2299; rev:2;)
```

Figura 2-6 Assinatura de acesso PHP

Mesmo se o tráfego é enviado no formato binário, e se houver um *payload* conhecido ou uma parte consistente do pacote (que é único para evitar falso positivos), que o NIDS possa detectar a detecção por *ruled-based* é possível. O tráfego da aplicação criptografado pode estar fora do alcance da maior parte dos NIDSs incapazes de detectar.

As tentativas de explorar campos da entrada introduzindo muitos dados, overflows conhecidos ou undeflows, que explorem falta de validação da entrada, e tentativas de injeção SQL são só algumas possibilidades que as aplicações específicas devem se preocupar. Naturalmente, as assinaturas variam muito dependendo da aplicação que está sendo protegida.

2.4.5 Informação Específica do Host

Enquanto a maior parte dos HIDSs não monitoram *tudo* o que acontece com o *host*, eles são capazes de avaliar todo o comportamento para um dado *host*, desde a criação de arquivos e acessos ao sistema à atividade de rede local na interface de

loopback. É muito comum para HIDSs criarem um banco de dados de estado do sistema (tamanhos de arquivo, permissões, tempos de acesso) quando eles são instalados, e então monitorem os desvios daquela *baseline*. De fato, para muitos tipos de HIDSs o processo de ajuste necessita da instalação do software HIDS, para então continuarem com a atividade de sistema normal e assim estabelecer uma *baseline* de modificação e quem o fez.

2.4.6 Informação Específica da Sub rede

A maior parte das redes tem padrões comuns de fluxos de tráfego. Se se souber que uma máquina na rede é um servidor de correio, não causa surpresa ao ver tráfego do *Simple Mail Transfer Protocol* (SMTP) que chega e sai do servidor. Quando se está acostumado com um dispositivo que monitora a rede executar um *ping* em cada dispositivo a cada cinco minutos, tal tráfego é aceitável, embora o mesmo comportamento proveniente de outro dispositivo de rede seja problemático. Dentro de algum tempo, um bom NIDS deve ser ajustado para reconhecer o comportamento esperado da subrede na qual ele reside, permitindo tráfego que é conhecido e aceitável, e enviando alertas para tráfego similares de *hosts* não autorizados. A estação de trabalho de um verificador de vulnerabilidades (*pent-tester)* autorizado pode vasculhar a rede, enquanto a estação de trabalho de um novo interno não pode.

Outro componente de tráfego que merece destaque é freqüentemente contemplado é o mapeamento da Camada 2. A Maior parte dos IDSs contemplam o tráfego *Address Resolution Protocol* (ARP), usado para fazer o mapa de endereços MAC na rede local. É possível que invasores imitem (spoof) o tráfego, modificando o endereço MAC ou forjando um endereço IP que não é o seu, e assim tentar interceptar o tráfego. Este tipo da falha pode ser visível em todos os níveis da sub-rede, dependendo da topologia da rede. Se o NIDS não estiver na mesma sub-rede local, na qual os ataques de Camada 2 estão acontecendo, ele não os descobrirá. Quando o tráfego de rede atravessa um roteador, o endereço MAC se modifica, e quando se verifica as portas e as posições dos endereços MAC, não se pode permitir que eles os modifiquem antes da verificação ou que os dados se tornem inseguros.

Por isso, quando se deseja capturar dados da Camada 2 com o NIDS, deve-se assegurar que o NIDS está na mesma sub-rede local com as demais máquinas que serão controladas, antes que qualquer roteador seja envolvido no *stream* de dados.

2.4.7 IDS Distribuído (Distributed)

Toda informação pode ser reunida e correlacionada com um DIDS, mas a escala é muito maior. Em vez de se usar a visão de rede local e da sub-rede e suas máquinas, utiliza-se uma visão da atividade através da empresa inteira. Pode-se escolher modelos de dados que seriam negligenciáveis ou sem importância em uma escala menor; e o que seria um *backup* automatizado de um servidor, resulta ser uma réplica coordenada (maliciosa) de dados por toda a rede, ao se analisar o quadro inteiro. Ver o tráfego do nível de DIDS permite observar fluxos de dados amplos e tendências totais mais claramente. O aspecto negativo é que se deve ter instrumentos para compreender efetivamente o montante de dados reunido; de outra maneira, os ataques sutis que se espera descobrir serão perdidos no meio do tráfego comum da empresa.

2.5 Como o IDS Observa Sua Rede

Sem um método efetivo de coleta de dados para análise, não existe nenhum propósito para o uso de um IDS. Felizmente, existem vários modos possíveis para o IDS coletar dados para analisar. A seguir serão mostrados os métodos mais comuns para a coleta de dados para o IDS. Cada um tem sua vantagem e desvantagem, e todos são bem apropriados para diferentes tarefas. Existem várias fontes possíveis de dados para um IDS.

2.5.1 Sniffing de pacotes

Qualquer IDS que analisa o tráfego da rede se utiliza de um "*sniffing"* de pacotes. Como foi mencionado, o NIDS opera pela configuração de uma interface no modo promíscuo e analisando pacotes nesta interface. Com isso, ele captura cada

pacote que atravessa a sub-rede local, e não irá notar os pacotes que passam pela pilha TCP/IP interna da máquina, porém potencialmente irá notar tudo no segmento local. Muitos HIDS que executam análises de tráfego de rede também usam técnicas similares sem o uso do modo promíscuo, para coletar o tráfego de um *host* no mesmo segmento. O s*niffing* de pacotes é um modo clássico de executar detecção de intrusão, e existem igualmente técnicas clássicas de evasão de IDS, que podem ser usadas contra um IDS fazendo *sniffing* de pacotes; por exemplo, ataques por fragmentação de pacotes, o qual divide o *payload* de ataque em vários pacotes. Como resposta tem-se a capacidade de o IDS reconstruir os pacotes, e então comparar os pacotes. O contra-ataque foi mudar o modo em que os pacotes são fragmentados, fazendo com que alguns dados se sobrescrevam, e novas técnicas para o IDS surgiram para essa possibilidade, criando um ciclo sem fim.

2.5.2 Log Parsing

Outra excelente fonte de dados de segurança são os arquivos de *log* do sistema. Muitos sistemas de IDS podem puxar dados de *log* de sistemas e alertar se eles detectarem qualquer coisa anômala. De fato, algumas das implementações originais de IDS usaram o monitoramento de *log* como seu método de coleta de dados. Alguns ataques são muito claros nos *footprint* que eles deixam no sistema e no *log*. O *Secure Shell* CRC32 *overflow*, por exemplo, pode deixar "*sshd[3698]: fatal: Local: crc32 compensation attack: network attack detected"* no seu log.

2.5.3 Monitoramento de Chamadas do Sistema

Os HIDS são capazes de se auto-configurarem como residentes no Kernel dos sistemas operacionais, e analisar (ou em alguns casos interceptar) chamadas potencialmente maliciosas do sistema. Uma chamada do sistema(system call) é uma requisição que um programa faz ao kernel do sistema operacional, se o HIDS acha que a chamada do sistema pode ser maliciosa, como uma requisição para troca de um ID de usuário para o de root, ele pode criar um alerta, ou no caso de alguns HIDS, como o *Linux Intrusion Detection System* (LIDS), não autorizar a chamada do sistema até que ocorra um cancelamento.

2.5.4 Observação do Sistema de Arquivo (Filesystem)

Outra tática muito comum para o HIDS é o tamanho e os atributos de arquivos cruciais do *filesystem.* Se o *kernel* do sistema operacional muda de tamanho e nenhum administrador do sistema sabe algo a respeito, isto provavelmente é algo para ser checado. Encontrar diretórios "*world-writrible"* ou encontrar binários comuns de sistema alterados é possível no caso de *Trojan*. Observando o *filesystem* desse modo, permite alertar os administradores contra possíveis atividades maliciosas; se não ao menos o mais breve possível. O *Tripwire* [10] é, talvez, o mais conhecido exemplo de ferramenta para monitorar mudanças em arquivos, mas existem muitos outros que fazem o mesmo serviço, incluindo a ferramenta *open-source Advanced Intrusion Detection Enviriomen*t (AIDE).

2.6 Como o IDS Captura Dado

2.6.1 Tentativas de Intrusão

Qualquer IDS irá coletar uma vasta quantidade de dados – redes estão ocupadas, servidores estão em pleno funcionamento, existe constantemente transferência de dados acontecendo, processos constantemente sendo executados. Para ser eficaz, um IDS deve ter pelo menos um (e possivelmente muitos) algoritmos para determinar qual tráfego merece a atenção do administrador. Existem muitas estratégias, porém existem duas opções táticas.

2.6.2 Saber o Bom ou Saber o Mau

O tráfego de rede pode ser identificado e classificado de muitas formas. Pode-se ter tráfego de acordo com uma dada política de segurança, ditada por necessidades particulares da empresa ou rede. Alguns administradores permitem apenas o tráfego que eles conhecem como bem, enquanto outros escolhem apenas bloquear o tráfego que eles desconhecem. Freqüentemente, aproximações baseadas em políticas irão centrar em uma aproximação do tráfego conhecido. Para tomar a melhor decisão para uma empresa, devem-se considerar quais os tipos de

tráfego deseja-se ter, quantas pessoas terão de lidar com o alertas, e o quão crítico pode-se ser.

Pode-se optar em identificar o tráfego conhecido como aceitável e marcar todo o restante, ou pode-se identificar os ataques conhecidos e deixar passar todo o tráfego restante. Neste contexto, temos o paradoxo básico da estratégia de um IDS; administradores de *firewall* estão bem familiarizados com este dilema. A estratégia do bem conhecido ou *know-good* irá ter ordens de magnitude de mais trabalho, tentando classificar todo o tráfego da rede, determinando o que supostamente aconteceria e o que é desejado. Desse modo, tem-se uma enorme quantidade de falsos positivos gerados por um IDS altamente ocupado, e isso irá selecioná-los lentamente para um nível gerenciável, que identificará o tráfego *know-good* na rede. Além disso, a menos que nada mude na rede, tem-se que constantemente afinar e refinar o IDS para ajustar às mudanças normais que acontecem com o tempo na maioria dos ambientes. Existem ferramentas automatizadas para definirem “normal”, onde “normal” é esperado como uma aproximação aceitável de “bom”. Entretanto, estas ferramentas sofrem com problemas de falsos positivos em ambientes complexos e altamente dinâmicos. Eles podem ainda ser enganados quando decidem que algo é “normal”, se a nova atividade ocorre em pequenas quantidades durante um longo período de tempo.

Entretanto, seguindo a estratégia de apenas alertar em tráfegos conhecidos ou suspeitos, irá resultar em um baixíssimo volume de alertas. Além do mais, devido às regras serem muito específicas na definição de algo ruim, quando um alerta (assumindo que as regras foram bem escritas), pode-se acreditar que a atividade “má” está sendo vista. Isto quer dizer que a pessoa que monitora o IDS não precisa ser habilidosa (pois não tem que fazer a verificação dos alertas ou *troubleshoot* do IDS). Contudo, com a aproximação há forte probabilidade do tráfego de ataque não combinar com as regras ou algoritmos. Com regras mais flexíveis, o número de falsos positivos pode aumentar. Em alguns cenários, não se conhece muito de detecção de intrusão e não se tem tempo de aprender, e isto pode não ser a melhor solução. Porém, se se deseja aumentar a probabilidade de detecção de um dado ataque, recursos disponíveis são necessários para monitorar e manter o IDS. A escolha da estratégia é uma análise de custo/benefício, pesando o tempo e o

recurso que se deseja dedicar aos IDS´s visando detectar o maior número de ataques.

2.6.3 Tecnologias para implementação da estratégia do IDS.

Diferenciar tráfego de ataque de atividades inócuas de rede e sistema em diversas formas. Alguns primeiramente usam uma técnica chamada análise *ruled-based* (também conhecida como *signature-based*), que compara um padrão conhecido com uma atividade vista no sistema ou rede. Foram apresentados exemplos de analisador de regras, a partir do conteúdo dos pacotes em uma rede, comparando-os com uma série de regras pré-definidas. A mesma coisa pode ser feita quando se observa as entradas dos arquivos de *log* ou de chamadas do sistema. Isto é similar ao modo como muitos programas anti-virus verificam as assinaturas de vírus no reconhecimento de blocos de arquivos infectados, programas, ou conteúdo ativo da Web ao entrar no sistema do computador (essa é a razão de se atualizar sempre o anti-virus). Detecção por assinatura é o método mais amplamente usado na tecnologia de IDS comerciais hoje em dia, pois é facilmente demonstrável, efetivo, e muito customizável com treinamento ou experiência limitada. Quando um novo ataque é descoberto, novas assinaturas podem ser escritas para comparar e alertar contra novas formas de ataque. Uma versão mais complexa da análise *ruled-based* é a análise de protocolo. Em vez de escrever uma simples regra que defina algo sobre um evento específico (bom ou mau), a análise de protocolo tenta definir todos os comportamentos aceitáveis possíveis para um tipo específico de atividades, Por exemplo, quando um computador deseja iniciar uma conexão TCP, ele envia pacotes SYN. As respostas aceitáveis são ambas RST/ACK ou SYN/ACK. Qualquer outra coisa pode ser uma violação de protocolo. Este método permite um pouco mais de flexibilidade na definição do que é “mau”. Em vez de dizer, “ao ver uma string maior que 500 bytes, preenchida com um caracter especifico, este é um ataque deste tipo”, pode-se dizer, “Neste ponto da conexão, não se deve ter *strings* maiores que 500 bytes. Se encontrar, isto é um ataque. Se ver mais de 500 bytes em outro ponto da conexão, tudo bem.” O problema é que, enquanto os protocolos são justos e bem definidos, nem todos os fabricantes prestam atenção em tudo na definição do protocolo. Como resultado, pode-se achar que IDS baseado em análise ou *analysis-based* está corretamente de

acordo com algo que não é permitido na RFC (*Request For Comments* – os documentos para definição da maioria dos protocolos da Internet. [ref rfc]), porém isto é completamente normal para aplicações de um fabricante específico. Além do mais, é tremendamente demorado e complexo escrever modelos de protocolos bons, e implementá-los de uma maneira suficientemente eficiente que possa ser usado na análise de redes de alta velocidade, pois isto leva anos de experiência. Significa que a maioria dos fabricantes tende a não ter muita vontade em compartilhar seus modelos de protocolos abertamente, mesmo para seus consumidores. Conseqüentemente, o "*troubleshooting"* de falsos positivos em IDS de análise de protocolo pode ser um longo processo dependente do fabricante. Outra técnica é denominada *anomaly detection*, que usa conceitos aprendidos ou pré-definidos sobre atividade do sistema: "normal" ou "anormal" (chamada heurística), para distinguir anomalias de comportamento normal do sistema; e monitorar, reportar ou bloquear anomalias assim que elas ocorram. Alguns IDS´s de *anomaly detection* vêm com padrões pré-definidos de como o tráfego normal deve parecer; já outros observam o tráfego na rede (ou atividades no sistema) e utilizam um algoritmo de aprendizado para desenvolver um perfil de "*baseline"* a partir daquilo. Estes perfis são *baselines* de atividades normais e podem ser construídos usando amostragem estatística, método *rule-based*, ou uma rede neural, para apenas citar alguns métodos.

Literalmente, centenas de fabricantes oferecem várias formas de implementações de IDS´s comercias. Devido à simplicidade de implementação, a maioria das implementações são primeiramente *signature based*, com poucas soluções de *protocol analysis* e limitadas às capacidades de detecção *anomaly-based,* presentes em certos produtos ou soluções.

2.7 O que o IDS faz Quando Encontra uma Tentativa de Ataque

Os IDSs mais modernos incluem capacidades de respostas automáticas limitadas, mas essas respostas normalmente concentram-se na filtragem de tráfego automatizada, bloqueio ou a desconexão como último recurso. Embora alguns sistemas sejam capazes de lançar contra-ataques, as melhores práticas indicam que a identificação automatizada e as facilidades de *backtrace* são os aspectos mais

úteis (e aqueles menos prováveis para processá-lo), que tais facilidades provêem e, por isso, são as mais usadas.

Há métodos diferentes e altamente configuráveis para o IDS executar quando descobre uma tentativa de intrusão. Na próxima seção é apresentado resumidamente os méritos de resposta ativa do IDS (às vezes erradamente conhecido como IPS, ou Sistemas de Prevenção de Intrusão) contra a detenção e alerta passivo mais tradicional.

2.7.1 Resposta Passiva (Passive Response)

Tradicionalmente, o IDS analisa a atividade, e pode ser configurado para registrar em um arquivo e/ou enviar alertas ao administrador. Estes alertas podem tomar muitas formas – armadilhas(TRAPs), Protocolo Simples de Gerência de Rede (SNMP), e-mails de saída, páginas ou mensagens de texto ao administrador do sistema, até chamadas telefônicas automatizadas. A maior parte dos administradores configuram o IDS para alertá-los de vários modos, dependendo da gravidade do ataque percebido e da freqüência da sua ocorrência. Não se deseja receber alertas por um pager 10 vezes por hora para algo que parece perigoso no início mas resulta em um falso positivo. Contudo, realmente a notificação de um alerta é esperado, especialmente se ele não indicar um falso positivo.

IDSs tradicionais limitam-se a essa tarefa. Eles são normalmente configurados com uma interface de gerência inteiramente separada de sua parte de escuta na rede, para não serem traídos por sua presença na parte que envia alerta todo o tempo. Muito freqüentemente, a interface de escuta não tem um endereço IP, e é uma interface invisível, configurada para não responder a qualquer tráfego.

2.7.2 Resposta Ativa (Active Response)

IDSs com capacidades de resposta ativas e IPSs (os dois são diferentes) emulam todo o comportamento de IDSs passivos tradicionais até a parte de detecção. Contudo, quando eles vêem uma tentativa de ataque, podem ser configurados para tomar medidas pró-ativas contra este, em vez de apenas alertar o administrador e esperar para uma tomada de decisão. Eles podem ser ligados em

série e eliminar o tráfego que vêem como malicioso, ou podem forjar o reset do Protocolo de Controle de Transmissão (TCP) com a fonte, ou com sistemas de destino (ou ambos) terminando abruptamente uma sessão TCP que vêem como tráfego de ataque. Podem ainda enviar mensagens de Protocolo de Mensagem de Controle de Internet (ICMP) do tipo destino inalcançável ao sistema fonte, como um esforço para convencê-lo que o sistema alvo é inalcançável. Alguns reconfiguram *firewalls* ou roteadores entre os alvos e os atacantes para bloquear o tráfego. Outros sistemas farão buscas de *nameserver* ou *traceroutes* no sistema de ataque, em uma tentativa de reunir a informação sobre este. Alguns fazem até *portscan* no sistema de ataque, e retornam um relatório de seu provável sistema operacional e possíveis vulnerabilidades.

A apelação de resposta ativa é por não ter um administrador de sistema monitorando a rede em tempo real. O perigo é a conseqüência da falta de configuração se tornar muito mais grave. O IPS deve ser verificado nas capacidades de "*whitelisting"* para se evitar cenários onde as aplicações perderam a conexão com o "*nameserver"*. É aconselhável ainda verificar as legalidades da jurisdição, na qual se planeja fazer com que o sistema automaticamente rastreie ou vasculhe os sistemas "atacantes".

2.7.3 IDS em linha (Inline)

Outro debate comum de configuração é se o IDS deve ficar em uma porta na rede *switched*, ou ficar ligado em série entre o *host* e a Internet. Há vantagens e desvantagens em ambas as configurações. Caso se pretenda ter o IDS agindo como um IPS, conectá-lo em série pode ser consideravelmente forte. A prevenção é muito mais eficaz quando o IDS é simplesmente capaz de descartar o tráfego que ele determinou como não permitido. Quando o IDS não é ligado em série, pode-se enviar um ICMP *unreachables,* ou TCP *Reset,* tanto à fonte como ao destino, mas se deve esperar que os próprios dispositivos se comportem apropriadamente. Como não se está controlando o segmento de rede entre eles, não há muita coisa que se possa fazer, com um IDS ligado em série, tem-se muito mais controle sobre estas situações.

Há duas preocupações principais neste tipo de configuração – os falsos positivos têm conseqüências mais desastrosas do que o IPS médio; e o desempenho pode ser um assunto significativo. Como todo o tráfego de rede está atravessando a caixa, um único ponto de perda é freqüentemente desastroso do ponto de vista do desempenho.

3 REDES NEURAIS ARTIFICIAIS(NEURAL NETWORS)

3.1 Introdução

As redes neurais artificiais têm sido baseadas em estudos de como o cérebro processa informações. Na sua forma mais geral, uma rede neural é uma máquina projetada para modelar a maneira como o cérebro realiza uma tarefa particular ou atividade de interesse. Uma rede neural é normalmente implementada utilizando-se componentes eletrônicos ou simulada por programação em um computador digital.

Segundo Haykim [10], uma rede neural é um processador maciçamente paralelo distribuído, constituído de unidades de processamento simples, que tem a propensão natural para armazenar conhecimentos experimentais e torná-los disponíveis para o uso, e assemelha-se ao cérebro em dois aspectos:

1. o conhecimento é adquirido pela rede por meio de dados do ambiente, num processo de aprendizagem. O Processo de treinamento é chamado de “Algoritmo de Aprendizagem”, que tem como finalidade ajustar os pesos sinápticos da rede de uma forma ordenada até alcançar um objetivo desejado.
2. as conexões entre os neurônios, conhecidas como pesos sinápticos, são utilizadas para armazenar o conhecimento adquirido.

Este capítulo apresenta um breve histórico de redes neurais artificiais. Apresenta o neurônio biológico e artificial, sendo este último a base de uma rede neural. Também os tipos de arquiteturas das redes e modelos de aprendizagem.

Apresenta também a arquitetura do Perceptron de Múltiplas Camadas (PMC) com retro propagação de erro, o qual é descrito de maneira geral o modo lote de treinamento, onde são apresentadas as derivadas e as fórmulas de ajuste do algoritmo. Também são descritas técnicas aplicadas no treinamento, como: (a) treinamento seqüencial e por lote; (b) taxa de momento; (c) validação cruzada; (d) curva de aprendizagem e heurísticas para melhorar o desempenho como classificador. Finalizando, apresentam-se as vantagens e as desvantagens do uso do Perceptron de Múltiplas Camadas.

3.2 Um Breve Histórico

Para alguns, o início das Redes Neurais Artificiais se deu em 1943 com o psiquiatra e neuroanatomista McCulloch e o matemático Pitts [11], que modelou o funcionamento de um neurônio. Em 1949 houve um avanço com a publicação do livro “The Organization of Behavior”, do neuropsicólogo Hebb [12], em que pela primeira vez foi apresentada uma regra de aprendizado fisiológico para as modificações sinápticas. Esta afirma que a eficiência de uma sinapse variável entre dois neurônios é aumentada pela ativação repetida de um neurônio, causado por outro neurônio, através daquela sinapse.

Em 1958, Rosenblatt [13] demonstrou o seu novo modelo: o perceptron, e em 1962 apresentou o “Teorema de Convergência do Perceptron”. Nesse período, muitos pesquisadores deram ótimas contribuições ao campo das redes neurais. Em 1960, Widrow e Hoff [14], baseados no método do gradiente para minimização do erro na saída de um neurônio com resposta linear, apresentaram uma regra de aprendizado conhecida como Regra de Widrow e Hoff ou Regra Delta. Mas, em 1969, Minsky e Papert [15] usaram uma matemática elegante para demonstrar a existência de limites fundamentais sobre o perceptron. Então veio a década da dormência.

Durante os anos 70 muitos pesquisadores, exceto os psicólogos e os neurocientistas, desistiram desse campo. Nos anos 80 ressurge o interesse pela Rede Neural Artificial, com a publicação de vários trabalhos como: Hop field (função de energia), Kohonen (selforganizing maps), Barto e Anderson (reinforcement learning). Em 1986, Rumelhart, Hinton e Williams [15] apresentaram a descrição do algoritmo de retro propagação de erro, mostrando que a visão de Minsky e Papert sobre o perceptron era bastante pessimista. Hoje, a Rede Neural Artificial representa uma vigorosa area de pesquisa multidisciplinar. E freqüentemente é identificada como uma subespecialidade da Inteligência Artificial; outras vezes como uma classe de modelos matemáticos para problemas de classificação e reconhecimento de padrões. E finalmente, numa categoria de modelos em ciência da cognição e conexionismo.

3.3 Neurônio Biológico e Artificial

O neurônio ou célula nervosa foi descrito por Ramon e Cajal, como sendo um dispositivo computacional elementar do sistema nervoso. O neurônio é formado por três regiões: o corpo celular, os dendritos e o axônio, como na Figura 3.1.

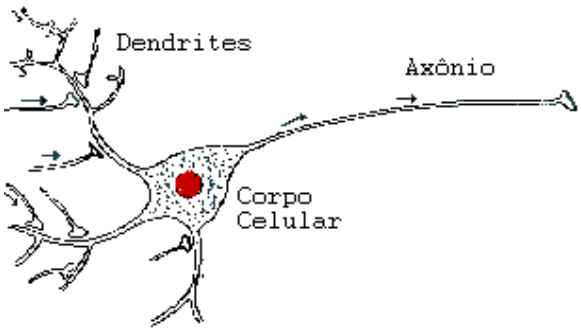

Figura 3-1 Neurônio biológico

- os dendritos, que tem formato de uma árvore, recebem as informações através dos longos axônios dos outros neurônios e as enviam para o corpo celular;
- o corpo celular, que recebe estímulos vindo dos dendritos, calcula se estes estímulos excedem o limiar de excitação ou “threshold” (i.e., ponto em que um

estímulo é suficiente para ser percebido ou produzir uma resposta) e a resposta é enviada para o axônio;

- o axônio, possui um longo conector cilíndrico que carrega o impulso de resposta do neurônio, que é transmitida para o dentrito neurônio seguinte.

O funcionamento do cérebro se dá através das comunicações entre os neurônios. Primeiramente os dendritos recebem os estímulos dos axônios de outros neurônios. Esses estímulos são enviados para o corpo celular que calcula o estímulo de saída do neurônio, que é transmitido pelo axônio para o dendrito de outro neurônio. A interligação entre os neurônios é chamada de sinapse e é feita através de substâncias conhecidas como neurotransmissores, que dependendo do resultado obtido no corpo celular podem ser excitatórias ou inibitórias. A comunicação no interior do neurônio se dá através de sinais elétricos, e entre neurônios por sinais químicos, Braga [16].

O cérebro humano possui aproximadamente 10 bilhões de neurônios no córtex cerebral e 60 trilhões de sinapses, Haykim [10].

Em 1943, o psiquiatra e neuroanatomista McCulloch e o matemático Pitts [11] foram os primeiros a proporem matematicamente uma forma simplificada de funcionamento do neurônio. No modelo de McCulloch, o neurônio possui i entradas (equivalente aos dendritos) $x_1, x_2, .., x_m$ e apenas uma saída (equivalente ao axônio), y. Para simular a sinapse, cada entrada do neurônio tem um peso atribuido $w_1, w_2, .., w_m$ cujos valores podem ser positivos (excitatórios) ou negativos (inibitórios). Os pesos têm como finalidade armazenar o conhecimento e determinar a intensidade com que cada entrada contribuirá no resultado final apresentado pelo neurônio. No corpo celular simplesmente é somando os valores do produto de suas entradas com seus respectivos pesos, $x_i\, w_i$, e se a soma for maior que o seu limiar a sua saída é ativada com valor 1, ou não ativada, com valor 0. Resumindo, o neurônio será ativo quando:

$$\sum_{i=1}^{m} x_i w_i \geq \theta \qquad \textbf{(3.1)}$$

onde m é o número de entrada do neurônio, w_i é o peso associado à entrada x_i, e é o limiar atribuído ao neurônio, conforme ilustrado na Figura 3.2.

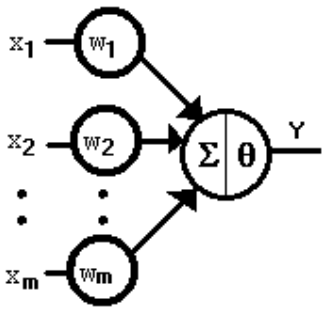

Figura 3-2 Neurônio artificial de McCulloch e Pitts

O neurônio de McCulloch e Pitts tem diferenças em relação ao neurônio biológico como: neurônios artificiais trabalham sincronamente, isto é, são ativados ao mesmo tempo; já os neurônios biológicos não possuem este tipo de sincronização. Nos sistemas biológicos, a saída do neurônio depende das ativações anteriores, devido aos neurotransmissores liberados anteriormente e que levam algum tempo para se recombinarem; nos neurônios artificiais são usados pesos negativos para representar essa inibição.

As principais limitações do neurônio de McCulloch e Pitts são: ele foi proposto com pesos fixos, isto é, não ajustáveis; e só conseguem implementar funções linearmente separáveis.

3.4 Arquitetura de Rede

A arquitetura de uma rede neural artificial depende diretamente do problema que será tratado pela rede. Como parte da definição da arquitetura da rede tem-se: quantidades de camadas, números de neurônios em cada camada e tipo de conexão entre os neurônios[16].

Quanto ao número de camadas, pode-se ter:

1. **redes de camada única**. A forma mais simples de uma rede em camadas surge quando se tem uma camada de entrada que se projeta para a camada de saída, mas não vice-versa, como mostrado na Figura 3.3(a) e (d);
2. **redes com múltiplas camadas**. Redes com múltiplas camadas, Figura 3.3 (b) e (c), se distinguem de redes com camada única pela presença de uma ou

mais camadas ocultas. A função das camadas ocultas é extrair informações das amostras.

Quanto aos tipos de conexões entre os neurônios, têm-se:

1. feedforward ou acíclica. A saída do neurônio na *iésima* camada não pode ter entradas com neurônios em camadas de índice menor ou igual a i, como mostrado na Figura 3.3(a), (b) e (c);
2. feedback ou cíclica. A saída do neurônio na iésima camada tem entradas com neurônios em camadas de índice menor ou igual a i, como mostrado na Figura 3.3 (d).

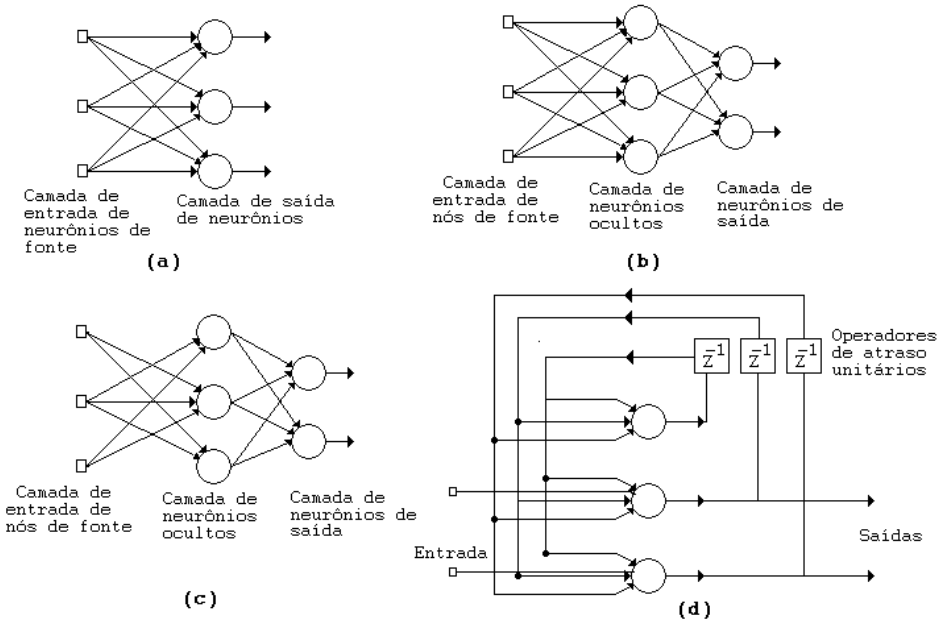

Figura 3-3 Exemplos de arquiteturas de RNAs

Finalmente, quanto a sua conectividade, tem se:

1. rede fracamente (ou parcialmente) conectada, como na Figura 3.3 (c);
2. rede completamente conectada, como mostrado na Figura 3.3 (a),(b)

e (d).

3.5 Processo de Aprendizagem

De importância primordial para as redes neurais é a habilidade de aprender a partir de seus ambientes e de melhorar o desempenho através do treinamento. Uma rede neural aprende acerca de seu ambiente através de um processo interativo de ajustes aplicados a seus pesos sinápticos e níveis de bias. Uma definição de aprendizagem, adaptada de Haykim [10] é: “Aprendizagem é um processo pelo qual os parâmetros livres de uma rede neural são adaptados através de um processo de estimulação pelo ambiente no qual a rede esta inserida”. O tipo de aprendizagem é determinado pela maneira com qual a modificação dos parâmetros ocorre.

Esta definição do processo de aprendizagem implica na seguinte seqüência de eventos:

1. a rede neural é estimulada por um ambiente;
2. a rede neural sofre modificações nos seus parâmetros livres (i.e. pesos sinápticos e bias) como resultado desta estimulação;
3. a rede neural responde de uma maneira nova ao ambiente, devido às modificações ocorridas na sua estrutura interna.

Como se pode esperar, não há um algoritmo de aprendizagem único para o projeto de redes neurais. Em vez disso, tem-se um conjunto de ferramentas representado por uma variedade de algoritmos de aprendizagem, cada qual oferecendo vantagens específicas. Basicamente, os algoritmos diferem entre si pela forma como são formulados os ajustes dos pesos sinápticos do neurônio. Outro fator a ser considerado é a maneira pela qual uma rede neural se relaciona com seu ambiente, que pode ser supervisionado ou não supervisionado, que será visto na seção 3.7.

3.6 Modelos de Aprendizagem em Relação ao Ajuste de Peso

Em 1949, Hebb propôs o princípio em que o aprendizado em sistemas nervosos complexos poderia ser reduzido a um processo local, com a intensidade

das conexões sinápticas alteradas apenas em função dos erros detectáveis localmente; isto significa que ao apresentar o n-ésimo exemplo, o ajuste em w_i depende unicamente do produto da iésima entrada pelo erro na saída. O padrão de treinamento é apresentado à rede, como na equação abaixo:

$$\omega_i^{novo} = \omega_i^{velho} + \eta(d_i - d_k)x_i$$

Essa é uma regra local, na medida detalhar os elementos que não depende dos valores das demais variáveis espalhadas pelo sistema.

Nas subseções que se seguem, apresentam-se modelos de aprendizagem.

3.6.1 Aprendizagem por Correção do Erro

Na Aprendizagem por Correção do Erro, a rede é estimulada por um vetor de entrada e por uma saída desejada. A regra de ajuste consiste em achar o erro subtraindo a resposta desejada da resposta da rede e calculando o gradiente descendente da função erro. O algoritmo implementado neste trabalho utiliza a Aprendizagem por Correção de Erro.

3.6.2 Aprendizagem Baseada em Memória

Na Aprendizagem Baseada em Memória faz-se uso da arquitetura de redes alimentadas com camada única e de modo supervisionado. A maioria das amostras de treinamento é armazenada em uma grande memória de padrões de entrada-saída. Um exemplo simples de aprendizagem baseada em memória é a regra do vizinho mais próximo, onde a rede classifica de acordo com a distância euclidiana entre os vetores de treino.

3.6.3 Aprendizagem competitiva

Na Aprendizagem Competitiva, após receber as características dos objetos da camada de entrada, os neurônios da camada de saída competem entre si, e o neurônio que tiver o maior valor discriminado se torna o vencedor. Enquanto em uma rede baseada na Aprendizagem Hebbiana vários neurônios na saída podem

estar ativos simultaneamente; na aprendizagem competitiva apenas um único neurônio na saída é ativado em um determinado instante. Essa característica torna a aprendizagem competitiva muito adequada para se descobrir características utilizadas para classificar um conjunto de objetos em determinadas classes.

3.7 Métodos de Aprendizagem

A escolha de um algoritmo de aprendizagem particular é influenciada pelo método de aprendizagem que uma rede neural deve executar [10]. Nesse contexto, identificam-se seis métodos de aprendizagem que se aplicam em redes neurais.

3.7.1 Associação de Padrões

Memória associativa é uma memória distribuída, inspirada no cérebro humano, que aprende por associação. A rede neural deve armazenar um conjunto de padrões que é apresentado repetidamente. Subseqüentemente, apresenta-se à rede uma descrição parcial ou distorcida de um padrão original armazenado, e a tarefa é recuperar aquele padrão particular. Um modelo de aprendizagem que pode ser usado neste método é a aprendizagem baseada em memória.

3.7.2 Reconhecimento de Padrões

O reconhecimento de padrões é o processo que, por meio das características do objeto, reconhece-se a qual classe o objeto pertence em um número pré-determinado de classes. Uma rede neural realiza o reconhecimento de padrões passando inicialmente por uma sessão de treinamento, durante o qual se apresenta repetidamente à rede um conjunto de características do objeto. Mais tarde, apresenta-se à rede um novo objeto que não foi visto antes, e a rede será então capaz de identificar a classe daquele objeto particular; por causa das informações que ela extraiu dos dados de treinamento. Nesse método, pode-se usar a aprendizagem por correção de erro.

3.7.3 Aproximação de Funções

Considere um mapeamento de entrada/saída não linear, descrito pela relação funcional:

$$d = f(x) \tag{3.2}$$

onde o vetor x é a entrada e d é a saída. Supõe que a função $F(.)$ seja desconhecida. Para compensar a falta de conhecimento sobre a função $f(\cdot)$ é fornecido um conjunto de exemplos rotulados:

$$T = \{(x_i, d_i)\}_{i=1}^{n} \tag{3.3}$$

O objetivo é projetar uma rede neural que aproxime a função desconhecida $f(\cdot)$ de forma que a função $F(\cdot)$ que descreve o mapeamento de entrada/saída realizado pela rede esteja suficientemente próxima à $f(\cdot)$, para todas as entradas, como:

$$\left\| F(x) - f(x) \right\| < \in \; para\; todo\; x \tag{3.4}$$

onde E é a tolerância atribuída ao erro, é um número positivo pequeno. Contanto que o tamanho n do conjunto de treinamento seja suficientemente grande e que a rede esteja equipada com um número adequado de parâmetros livres, então o erro pode ser suficientemente pequeno. Nesse método pode-se usar a aprendizagem por correção de erro.

3.8 Perceptron

Utilizando o modelo de neurônio de McCulloch e Pitts [11], Frank Rosenblatt [13] escreveu em 1958 o primeiro conceito de aprendizado em RNAs, e demonstrou o Teorema de Convergência do *Perceptron*; em que o algoritmo de aprendizado do *perceptron* sempre converge, caso o problema em questão seja linearmente separável.

O aprendizado ou adaptação tem como finalidade calcular o valor de $\Delta\omega$ a ser aplicado nos pesos ω, que tende a encontrar o melhor ω que resolve o problema em questão.

Considerando um neurônio arbitrário da camada de saída de um *perceptron,* com vetores de entrada x_0 e pesos ω_0, com ativação igual ao produto interno de ω e x, ∑ ωi xi, isto é, o angulo entre ω e x.

$$\sum w_i x_i = \left\{ \begin{array}{l} = 0, \omega \perp x \\ > 0, g < 90^\circ \\ \leq 0, g > 90^\circ \end{array} \right\}$$

A condição de ativação do neurônio ocorre quando ∑ω'x' = θ, onde θ é o valor de limiar do neurônio; fazendo ∑ω'x'− θ = 0, para simplificar, basta adicionar no vetor de entrada, x(0) = 1, e no vetor peso, ω(0) = −θ.

Utilizando {x, d} como par de treinamento, sendo x o vetor de entrada, d a resposta desejada de x e y a resposta produzida pela rede. O erro do neurônio será $e=d-y$ para o vetor x. Os valores de y e d no caso do *perceptron* podem ser y Є {0, 1} e d Є {0, 1}, portanto têm-se duas situação para $e \neq 0$:

1. a primeira é quando $d = 1$ e $y = 0$, neste caso $e = 1$ e ∑ω'x' < 0, que implica o ângulo entre ω' e x' ser $g > 90$. De acordo com a Figura 3.4 (a), uma boa opção para a alteração do vetor peso é somar o vetor ηx. Assim, $\Delta w = \eta x$ e $\omega(t + 1) = \omega(t) + \eta x$, como $e = 1$, pode-se ter $\Delta w = \eta e x$ e $w(t + 1) = w(t) + \eta e x$, onde η é a taxa de aprendizado do algoritmo, isto é, o quanto o vetor peso irá ser modificado;
2. a segunda é quando $d = 0$ e $y = 1$, neste caso $e = -1$ e $\sum \omega' x' \geq 0$, que implica o ângulo entre ω' e x' ser < 90 . De acordo com a Figura 3.4 (b), uma boa opção para a alteração do vetor peso é subtrair o vetor ηx. Assim $\Delta\omega = -\eta x$ e $\omega(t + 1) = \omega(t) - \eta x$, como $e = -1$ pode-se ter $\Delta\omega = \eta e x$ e $\omega(t + 1) = \omega(t) + \eta e x$.

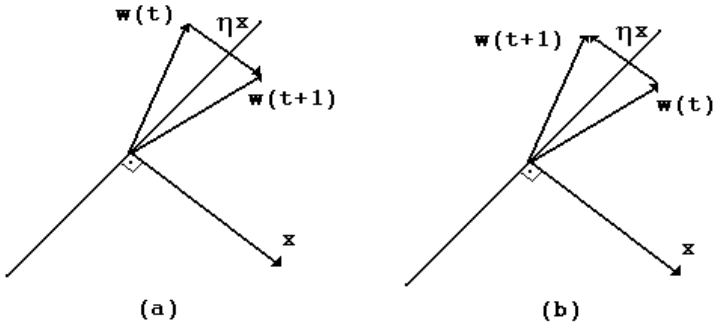

Figura 3-4 Unidade de saída

Assim, finalmente tem-se a equação 3.5:

$$\omega_{(t+1)} = \omega_{(t)} + \eta e x \tag{3.5}$$

3.9 *Perceptron* de Múltiplas Camadas

Em 1986, Rumelhart, Hinton e Williams [15] apresentaram a descrição do Algoritmo Retropropagação de erro ou *Backpropagation* para arquitetura do *perceptron* de Múltiplas camadas, mostrando que a visão de Minsky e Papert sobre o *perceptron* era bastante pessimista. Os perceptrons de Múltiplas camadas com retropropagação de erro têm sido aplicados com sucesso para resolver diversos problemas. Como exemplo: reconhecimento de caracteres, previsão do comportamento de ações na bolsa, verificação de assinaturas, segurança em transações com cartões de crédito, diagnóstico médico e outros [16].

3.9.1 Arquitetura do Perceptron de Múltiplas Camadas

Utilizando como estrutura o neurônio artificial como o da Figura 3.5, tem-se a base para projetos de redes neurais artificiais, onde se pode observar três elementos básicos:

1. entrada, representado pelo produto interno de x_m e w_{km} . Assim, um sinal x_j na entrada da sinapse j conectado ao neurônio k e multiplicado pelo peso

sináptico w_{kj}. O primeiro índice se refere ao neurônio em questão e o segundo se refere ao terminal de entrada à qual o peso se refere;

2. um somador, o produto interno dos sinais de entrada com os respectivos pesos do neurônio;
3. uma função de ativação, ou de restrição, que restringe a amplitude da saída y_k do neurônio.

O modelo neural da Figura 3.5, possui também bias, aplicado externamente, representado por $b\boldsymbol{k}$. O bias bk tem o efeito de aumentar ou diminuir a entrada líquida da função de ativação, dependendo se ele é positivo ou negativo, respectivamente.

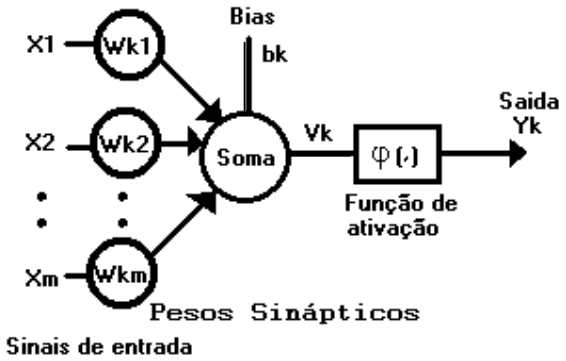

Figura 3-5 Neurônio artificial

Utilizando como estrutura básica da rede o Neurônio da Figura 3.5, tipicamente uma MLP ou PMC possui uma camada de entrada, uma ou mais camadas ocultas e uma camada de saída. A Figura 3.6 mostra a arquitetura de uma rede *Perceptron* de Múltiplas Camadas, com uma camada de entrada, uma oculta e uma de saída. Note que um neurônio em qualquer camada da rede esta conectado a todos os neurônios da camada anterior.

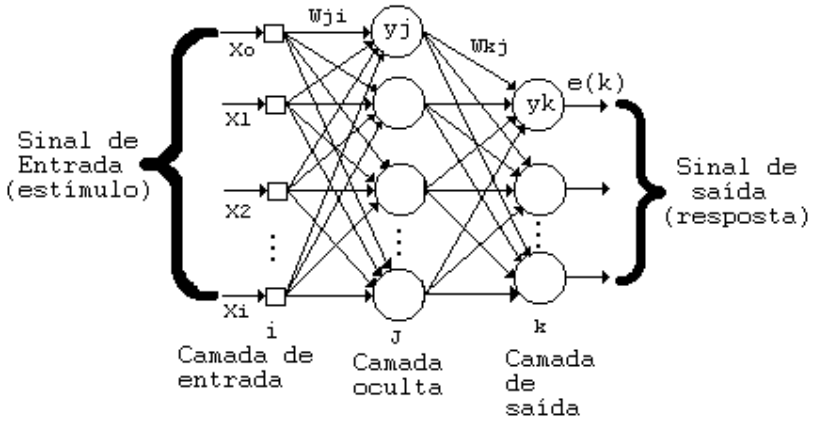

Figura 3-6 Arquitetura do *Perceptron* de Múltiplas Camadas com uma camada oculta

3.9.2 O Algoritmo de Retropropagação

O algoritmo de retropropagação de erro possui duas fases distintas:

1. para frente ou propagação (em inglês *forward*). As características de uma amostra são introduzidas na camada de entrada, propaga-se para frente (neurônio por neurônio) através da rede e emerge na camada de saída da rede;
2. para traz ou retropropagação (em inglês *backward*). O erros que se originam nos neurônios da camada de saída, se propagam para trás (camada por camada) por meio da rede, atualizando os seus pesos.

A Figura 3.7, representa os dois tipos de fluxos identificados nesta rede.

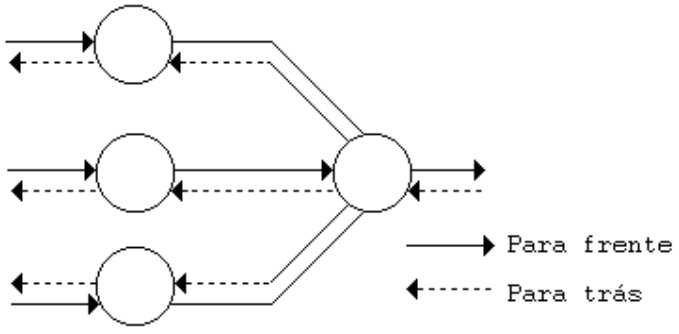

Figura 3-7 Ilustração das direções de dois fluxos básicos em um *Perceptron* de Múltiplas Camadas: propagação para frente e a retropropagação.

3.9.3 Modo de Treinamento Seqüencial e por Lote

Como mencionado anteriormente, uma apresentação completa do conjunto de treinamento inteiro é denominado uma época. Para um dado conjunto de treinamento, a aprendizagem por retropropagação pode então proceder de formas básicas:

1. modo seqüencial. O modo seqüencial de aprendizagem por retropropagação é também chamado de modo *online*, modo padrão ou modo estocástico. Neste modo de operação, a atualização dos pesos é realizada após a apresentação de cada padrão de treinamento; este é o modo de apresentação onde se aplica o algoritmo de retropropagação apresentado.

   Para ser específico é preciso considerar uma época constituindo de N padrões (vetores) de treinamento arranjados na ordem (x_1 , d_1), .., (x_N , d_N). O primeiro par de padrão (x_1, d_1) da época é apresentado à rede; e a seqüência de computações, para frente e para trás, descrita anteriormente, é realizada resultando em certos ajustes dos pesos sinápticos e níveis de bias da rede. Então, o segundo par de padrões (x_2 , d_2) da época é apresentado, e a seqüência de computações para frente e para trás é repetida, resultando em novos ajustes dos pesos sinápticos e níveis de bias. Esse processo continua até que o ultimo par de padrões (x_N , d_N) da época seja considerado;

2. modo por lote: No modo por lote a aprendizagem por retropropagação o ajuste dos pesos é realizado após a apresentação de todos os padrões de treinamento que constituem uma época.

Do ponto de vista operacional *online*, o modo seqüencial de treinamento é preferível em relação ao modo por lote, porque requer menos armazenamento local para cada conexão sináptica. Além disso, dado que os parâmetros são apresentados à rede de uma forma aleatória, o uso de ajuste de pesos de padrão em padrão torna a busca no espaço de pesos puramente aleatória. Isso torna menos provável que o algoritmo de retropropagação fique preso em um mínimo local [10].

O modo de treinamento por lote fornece uma estimativa precisa do vetor gradiente. A convergência para um mínimo local é assim garantida sob condições

simples. A composição do modo por lote também o torna mais fácil de ser paralelizado do que o modo seqüencial.

Quando os dados de treinamento são redundantes (i.e., o conjunto de dados contém várias cópias de padrões), constata-se que, diferentemente do modo por lote, o modo seqüencial é capaz de tirar vantagens de sua redundância, porque os padrões são apresentados um de cada vez. Isso ocorre particularmente quando o conjunto de dados é grande e altamente redundante [10].

3.9.4 Vetor Gradiente

Admitindo que uma função f : $D \subset R^2 \rightarrow R$, diferençável em P_0D, define-se o vetor gradiente da função f no ponto P_0 por $\nabla f\ (P_0)$ ou $gradf(P_0)$ ao vetor cujas componentes são as derivadas parciais de f no ponto P_0.

Uma importante propriedade do gradiente é que ele aponta na direção de maior crescimento de f em (x, y). O vetor oposto ao gradiente (gradiente descendente) é dado por $-\nabla f(x,y) = \left(-\frac{\partial f}{\partial x}(x,y), -\frac{\partial f}{\partial y}(x,y) \right)$, e aponta na direção em que a função decresce mais rapidamente.

$$\nabla f(P_0) = \left(\frac{\partial f(P_0)}{\partial x}, \frac{\partial f(P_0)}{\partial y} \right) \tag{3.6}$$

Como exemplo de cálculo do vetor gradiente da função $f(x, y) = x^2 + y^2$, com $z = f(x, y)$, e $(x, y) \in \Re^2$, tem-se:

$$\nabla f(x,y) = \left(\frac{\partial f}{\partial x}(x,y), \frac{\partial f}{\partial y}(x,y) \right) = (2x, 2y)$$

Fazendo $x_0 = 1$ e $y_0 = 1$, obtém:

$$\nabla f(1,1) = (2,2)$$

A Figura 3.8 ilustra o vetor gradiente no ponto (x_0, y_0).

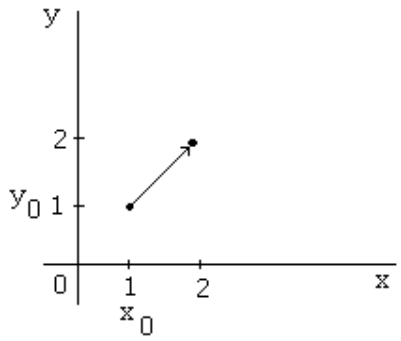

Figura 3-8 Vetor gradiente do ponto f (x0, y0)

Fazendo η = 0.1, o ponto (x_1 , y_1) pode ser obtido utilizando o gradiente:

$$(x_1, y_1) = (x_0, y_0) + \eta \nabla f(x_0, y_0)$$

$$(x_1, y_1) = (1,1) + 0.1(2,2) = (1.2,1.2)$$

O gradiente para o ponto (x_1, y_1) é:

$$\nabla f(x_1, y_1) = (2x_1 , 2y_1) = (2.4, 2.4)$$

Do mesmo modo, o ponto x_2 e y_2 :

$$(x_2, y_2) = (x_1, y_1) + \eta \nabla f(x_1, y_1)$$

$$(x_2, y_2) = (1.2,1.2) + 0.1(2.4,2.4) = (1.44,1.44)$$

Finalizando, os pontos:

$$z_0 = f(x_0 , y_0) = 2,\ z_1 = f(x_1 , y_1) = 2.88 \text{ e } z_2 = f(x_2, y_2) = 4.14$$

A Figura 3.9 ilustra a caminhada do gradiente para a maximização da função *f*. Esse procedimento deve ser repetido até que seja alcançado um valor máximo para z.

A função EQM que se quer minimizar depende dos pesos ws. Para uma rede com 2 entradas, 3 neurônios na camada intermediária e 2 neurônios na camada de saída, a EQM estaria definida em R^{17}, o que torna impossível a representação geométrica.

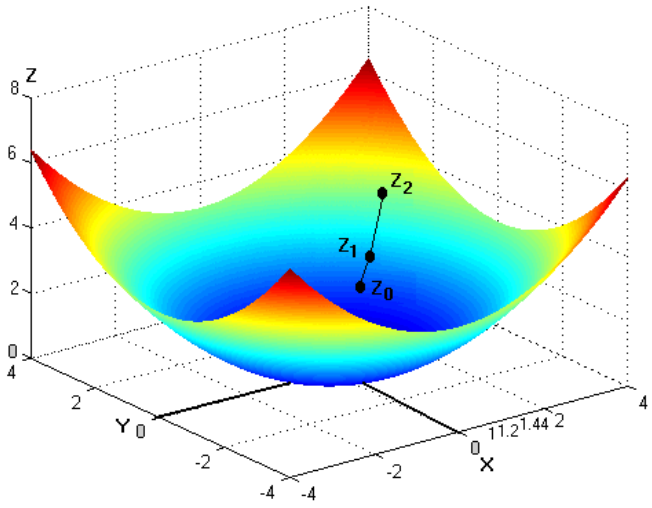

Figura 3-9 Gráfico da função f (x, y) = x2 + y2

3.9.5 Momento

O algoritmo de retropropagação fornece uma aproximação para a trajetória no espaço de pesos calculada pelo método da descida mais íngreme. Quanto menor for o parâmetro da taxa de aprendizagem η, menor será a variação dos pesos sinápticos da rede de uma iteração para a outra, e mais suave será a trajetória no espaço de pesos. Essa melhoria, entretanto, é obtida à custa de uma taxa de aprendizagem lenta. Por outro lado, se o parâmetro da taxa de aprendizagem η for muito grande, grandes modificações nos pesos sinápticos resultantes podem tornar a rede instável (i.e., oscilatória). Um método simples de aumentar a taxa de aprendizagem, evitando, no entanto, o perigo de instabilidade, é modificar a regra delta, incluindo um termo de momento, como:

$$\Delta\omega_{ji}(n) = \alpha\Delta\omega_{ji}(n-1) + \eta\delta_j(n)y_i(n) \qquad (3.7)$$

onde α é usualmente um número positivo chamado de constante de momento.

A incorporação do momento no algoritmo de retropopagação representa uma modificação pequena na atualização dos pesos; contudo, pode haver alguns efeitos benéficos sobre o comportamento de aprendizagem do algoritmo.

O termo de momento também pode ter o efeito de evitar que o processo de aprendizagem termine em um mínimo local.

3.10 Validação Cruzada

A validação cruzada ou *cross validation* tem como princípio ajudar na parada do treinamento. Primeiramente, divide-se o conjunto de dados aleatoriamente em um conjunto de treinamento e um conjunto de teste. O conjunto de treinamento é dividido adicionalmente em dois subconjuntos disjuntos:

a) Um subconjunto de estimação, usado para o ajuste dos parâmetros da rede;

b) Um subconjunto de validação, usado para testar ou validar o modelo.

O subconjunto de estimação é usado para treinar a rede na maneira usual, exceto por uma pequena modificação: a sessão de treinamento é interrompida periodicamente (i.e., após um número determinado de épocas), e a rede é testada com o subconjunto de validação: isto é, após um período de estimação (treinamento), os pesos sinápticos e os níveis de bias da rede são todos fixados e a rede atua em seu modo direto. O erro de validação é então medido para cada exemplo do subconjunto de validação. Quando a fase de validação é completada, a estimação (treinamento) é reiniciada em um novo período, e o processo é repetido.

A Figura 3.12 mostra as formas conceituais de duas curvas de aprendizagem: uma relativa às medidas sobre o subconjunto de estimação, e a outra relativa ao subconjunto de validação. A curva de aprendizagem de estimação decresce para um número crescente de épocas. Diferentemente, a curva de aprendizagem de validação decresce para um mínimo e então começa a crescer conforme o treinamento continua. Quando se olha para a curva de aprendizagem da estimação pode parecer que pode se melhorar o desempenho, indo além do ponto mínimo da curva de aprendizagem de validação. Entretanto, na realidade, o que a rede aprende após esse ponto é essencialmente o ruídos contidos nos dados de treinamento. Essa heurística sugere que o ponto mínimo na curva de aprendizagem de validação seja usado como critério sensato para encerrar a sessão de treinamento. Esse procedimento é referido como método de treinamento com parada antecipada.

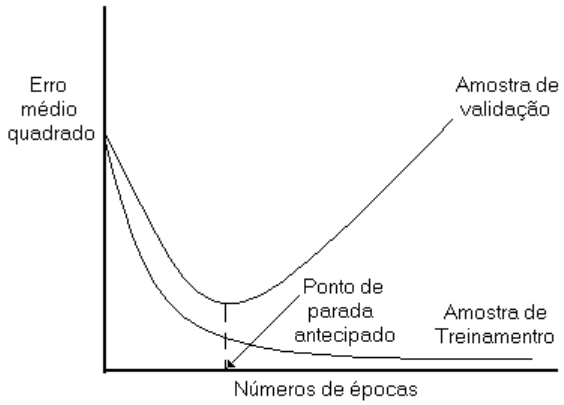

Figura 3-10 Ilustração da regra de parada antecipada baseada na validação cruzada

3.11 Generalização

A rede neural utiliza o conjunto de treinamento para adaptar os pesos e bias da rede, de forma que a rede neural seja capaz de generalizar. Uma rede generaliza bem quando o mapeamento de entrada e saída computado pela rede for correto para dados de teste que não foram utilizados para o treinamento da rede. Assume-se que os dados de teste são retirados da mesma amostra usada para gerar os dados de treinamento.

Uma rede neural pode ter uma generalização pobre, quando há neurônios ocultos mais do que o necessário, resultando em memorização. De outro modo, com pequeno número pode ocorrer *underfitting*, onde a rede não consegue convergir durante o treinamento.

Considerando a rede neural como um mapeamento não linear de entrada e saída, ilustrada na Figura 3.13a, onde os pontos rotulados como “dados de treinamento” foram utilizados na aprendizagem e o ponto marcado como “generalização” tem-se o resultado da interpolação realizada pela rede. A rede pode ter generalização pobre devido à memorização, como ilustrada na Figura 3.13b para os mesmos dados utilizados na Figura 3.13a. A memorização implica que o mapeamento de entrada e saída da rede não é suave [10].

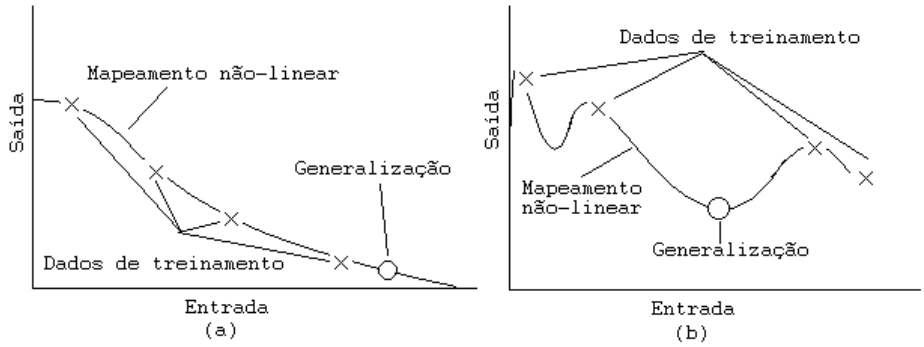

Figura 3-11 (a) Dados ajustados adequadamente; (b) Dados ajustados em excesso

3.12 Heurística para Melhorar o Desempenho

Freqüentemente é dito que o projeto de uma rede neural utilizando o algoritmo de retropropagação é mais uma arte do que uma ciência, significando que muitos dos numerosos fatores envolvidos no projeto são resultados da experiência particular de cada um. Há alguma verdade nessa afirmação, entretanto, existem métodos que melhoram significativamente o desempenho do algoritmo de retropropagação, como descrito a seguir:

1. Aleatorizar o conjunto de treinamento. Nas tarefas de classificação de padrões usando a aprendizagem por retropropagação seqüencial, uma técnica simples e bastante utilizada é tornar aleatória a ordem em que os exemplos são apresentados ao *perceptron* de Múltiplas camadas entre épocas. Idealmente, a aleatoriedade garante que os exemplos apresentados à rede em uma época raramente pertençam à mesma classe;
2. Valores-alvo. É importante que os valores alvo (resposta desejada) sejam escolhidos dentro do intervalo da função de ativação sigmóide. Mais especificamente, a resposta desejada d_j para o neurônio j na camada de saída do *perceptron* de Múltiplas camadas deve ser deslocada por uma quantidade ε do valor limite da função de ativação sigmóide, dependendo se o valor limite é positivo ou negativo. Caso contrário, o algoritmo de retropropagação tende a levar os parâmetros livres da rede para o infinito e

dessa forma reduzir a velocidade do processo de treinamento, levando os neurônios ocultos à saturação;

3. Normalizar as entradas. Cada variável de entrada deve ser pré processada, de modo que o seu valor médio, calculado sobre todo o conjunto de treinamento, seja próximo de zero, ou seja, pequeno quando comparado com o desvio padrão. Para avaliar o significado prático dessa regra, deve-se considerar o caso extremo, onde as variáveis de entrada são positivas de modo consistente. Nessa situação, os pesos sinápticos de um neurônio, na primeira camada oculta, podem apenas crescer juntos ou decrescer juntos. Conseqüentemente, se o vetor peso daquele neurônio deve mudar de direção, ele só pode fazer isso ziguezagueando em seu caminho por meio da superfıcie de erro, o que é tipicamente lento e deve ser evitado.

   Para acelerar o processo de aprendizagem por retropropagação, a normalização das entradas deve incluir também duas medidas: (a) as variáveis de entrada contidas no conjunto de treinamento não devem ser correlacionadas. Isso pode ser feito utilizando-se da analise dos componentes principais; (b) as variáveis de entrada não correlacionadas devem ser escalonadas para que suas covariâncias sejam aproximadamente iguais, assegurando-se com isso que os diferentes pesos sinápticos da rede aprendam velocidades próximas;

4. Variar o número de neurônios na camada oculta;
5. Escolher uma taxa de aprendizado pequena para não ocorrer *overshooting* da solução (ou seja, ir além e perder a solução);
6. A taxa de aprendizado é escolhida experimentalmente para cada problema;
7. A inicialização afeta muito o resultado final. Reinicializações podem resolver este problema.

3.13 Vantagens e Desvantagens

De acordo com Haykim [10], há vantagens e desvantagens no algoritmo retropropagação de erro:

a) Vantagens:

O cálculo localizado das unidades neuronais permite manter o desempenho na presença de erros, portanto, fornece uma base de projeto tolerante a falhas.

A rede neural por ter natureza paralela pode ser distribuída implementada utilizando a tecnologia VLSI (*Very Large Scale Integration*). A complexidade computacional do Algoritmo *Backpropagarion* é polinomial em relação ao ajuste dos parâmetros (pesos) que devem ser alterados de uma iteração para a iteração seguinte. Nesse sentido, o Algoritmo *Backpropagarion* é computacionalmente eficiente.

b) Desvantagens:

O algoritmo de retropropagação de erro é basicamente uma técnica de escalada de encosta. A existência de mínimos locais (vales) pode gerar uma falsa solução ótima, ou aumentar o tempo de treinamento até sair do vale.

Os problemas de treinamentos em grande escala de redes neurais em são considerados difíceis, e não existe uma estratégia única de aprendizagem, sendo necessárias outras abordagens, como por exemplo, o uso de pré-processamento, o uso de eficiente, algoritmos genéticos, e outras abordagens de IA.

4 MODELO E METODOLOGIA

Para se efetuar os teste, optou-se por uma base de dados(*datasets*) que fosse capaz de simular vários tipos de ataques, bem como tráfego normal de uma rede. A composição do *dataset* deveria possuir padrões de ataques bem como padrões de tráfego normal, para que a rede neural possa diferenciar um do outro e não apenas identificar algum ataque.

Ao procurar métodos de avaliação do modelo proposto, encontramos outros estudos com resultados satisfatórios e que utilizaram a base de dados do DARPA 98 [17], que foi criada no MIT, o que nos deu uma direção para o desenvolvimento do método. Prosseguindo na busca, encontrou-se um outro *dataset* que também foi utilizado pelo DARPA, porém em 99[18], incorporando novos tipos de ataques, procurando deixa-lo mais completo. Neste mesmo ano, em uma competição de Data Minning , foi criado um *dataset* baseado no DARPA 98 e 99, com o intuito de testar algorítimos de extração de dados em um *dataset*. Essa base de dado, em vez de ter um tamanho enorme como a do DARPA, tinha um resumo das conexões contidas no DARPA e as conexões estavam descritas em 41 campos que serão descritos mais adiante neste capítulo.

Tinha-se então que escolher entre 3 (três) bases de dados para aplicarmos ao modelo de Sistema de Detecção de Intrusão, as oriundas do DARPA possuíam todas as conexões criadas durante as 4 semanas de simulação de tráfego, bem

como todos os pacotes. Isso tinha gerado um arquivo de ordem de Giga Bytes de informação, que deveriam ser processadas.

Processar toda essa informação não seria o primeiro obstáculo, pois antes de processar a informação deveria-se colocá-la em um formato conhecido, pois como a base da dados foi gerada a partir de um sniffer, tinha-se que abrir os arquivos novamente com o sniffer, para que ele pudesse gerar os pacotes novamente e envia-los pela rede e assim processar os pacotes diretamente da rede.

Ao analisar o *dataset* gerado pelo KDD Cup em 1999[19], o trabalho de desempacotar os arquivos gerados pelo sniffer e o primeiro processamento dos pacotes gerados pelos arquivos, já estava feito, faltava apenas aplicar o modelo.

Optou-se então pela utilização do *dataset* gerado pelo KDD, o que aceleraria o processo de avaliação do modelo e teria mais tempo para a modelagem do conhecimento e o treinamento da rede neural, procurando obter o melhor resultado possível do treinamento.

O *dataset* é composto por padrões de invasões(ataques Remote-to-local e User-to-Root), também conhecidos como *exploits*, padrões de reconhecimento de vulnerabilidades (Probes) e ataques de negação de Serviço (DoS). Juntamente com padrões de tráfego normais.

Todos os padrões estão com etiquetas (label) informando qual é o tipo de tráfego que ele pertence, facilitando o treinamento e também a análise de resultados do modelo. A seguir veremos em detalhes o que são os exploits que compõem a base de dados DARPA 98, que é a origem do *dataset* KDD CUP que estaremos utilizando para avaliação de desempenho do modelo de Sistema de Detecção de Intrusão. Veremos também como foi separado em classes os tipos de ataques e quais critérios foram usados para essa separação.

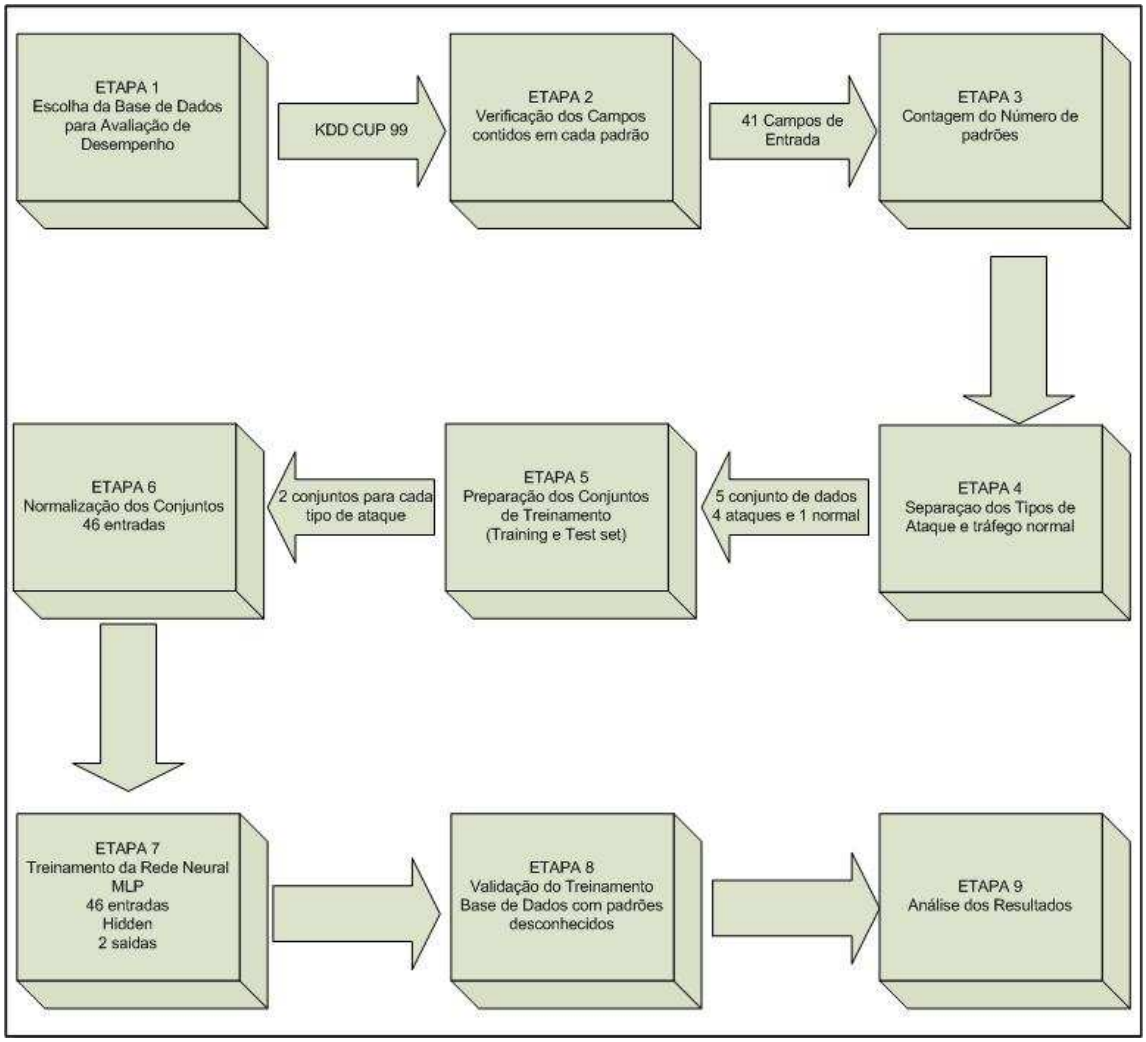

Figura 4-1 Diagrama em Blocos das Atividades Realizadas

4.1 Exploits

Uma grande amostra de ataques reais de computador é necessária para testar um sistema de detecção de intrusão. Esses ataques devem cobrir as diferentes classes de tipos de ataque e conter "*exploits"* tanto para vulnerabilidades recentemente descobertas, como para as mais antigas. Um exemplo de ataque deve consistir de várias fases. Por exemplo, um atacante poderia copiar um programa para a um sistema, executar este programa que explora uma vulnerabilidade do sistema para ganhar privilégios de *root* e, então, usar este privilégio de *root* para instalar um "*backdoor"* no sistema para um acesso posterior [20].

4.1.1 Fontes

Muitos dos *exlpoits* desenvolvidos para a avaliação, DARPA 1998, foram feitas de idéias ou implementações disponíveis de fontes públicas na Internet. O "*Rootshell*" [21] é um site web dedicado à reunião de *exploits* de computador e tem um arquivo relativamente grande de ataques para muitos sistemas operacionais populares. O *"Bugtraq"* também freqüentemente armazena "*exploit code*" - ostensivamente lançado com o objetivo de testar as vulnerabilidades do próprio sistema de alguém - quando uma nova vulnerabilidade é discutida. Um arquivo de pesquisa do Bugtraq pode ser encontrado na Internet em http://www.geek-girl.com. Outros exploits foram criados da informação lançada por grupos de segurança como CERT [22] e X-force ISS [23] que freqüentemente lança informação sobre novas vulnerabilidades. Fontes de informações adicionais sobre vulnerabilidades de sistemas e os possíveis exploits são boletins dos fabricantes como a Sun Microsystems e a Software Redhat. Esses boletins são lançados para os clientes a fim de estimulá-los a carregar os *pacthes* que eliminam essa nova vulnerabilidade. Os novos *exploits* foram criados especificamente para o objetivo de avaliação do IDS. Esses novos *exploits* são úteis para se determinar quão bem um sistema de detecção de intrusão trabalha contra novos ataques, que não eram publicamente conhecidos no tempo em que o sistema de detenção de intrusão foi desenvolvido.

4.1.2 Idade do Exploit

Cada novo *exploit* tem um período de tempo durante o qual é mais perigoso, como pode-se ver na figura 4-3. Com o passar do tempo, mais pessoas tomam conhecimento da vulnerabilidade e aplicam os *patches* (remendos) aos seus sistemas para fazê-los resistente ao *exploit*. Mesmo depois das notícias de uma vulnerabilidade tornar-se comum, alguns sistemas podem não estar preparados. Alguns administradores de sistemas com menos experiência passam anos sem ter falhas de segurança populares detectados e consertados. Alguns desses ataques mais velhos foram incluídos no sistema de ataques usados para a avaliação DARPA 1998.

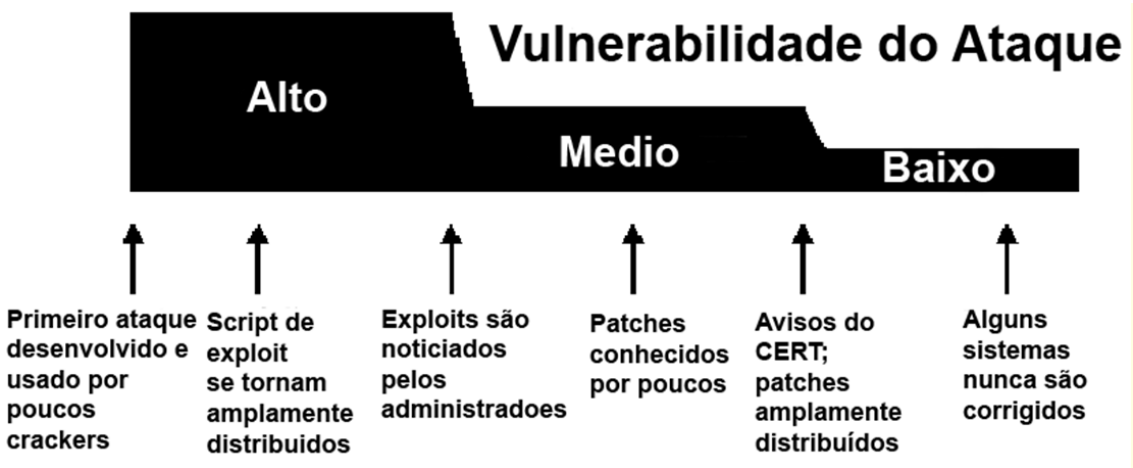

Figura 4-2 - Vulnerabilidade do Ataque

4.1.3 Taxonomia de Ataques de Computadores

Uma taxonomia para classificar ataques de computador foi usada para escolher os *exploits*. Uma boa taxonomia permite classificar ataques em grupos, com propriedades em comum. Uma vez que esses grupos foram identificados, o trabalho de testar apropriadamente um sistema de detenção de intrusão torna-se mais fácil, porque em vez de desenvolver cada possibilidade ataque, pode-se escolher um subconjunto representativo em cada grupo. A taxonomia apresentado aqui foi originalmente apresentada em [24]. As características desta taxonomia são:

- Cada ataque pode ser confiadamente colocado em uma categoria;
- Todas as intrusões possíveis têm um lugar na taxonomia;
- A taxonomia pode ser estendida no futuro;

Esta taxonomia foi criada com o objetivo de provar e avaliar sistemas de detecção de intrusão, e cada ataque pode ser categorizado como um dos seguintes:

- Um usuário executa alguma ação em um nível de privilégio;
- Um usuário faz uma transição não autorizada de um nível de privilégio mais baixo para um nível mais alto nível de privilégio.

- Um usuário fica no mesmo nível de privilégio, mas executa alguma ação de um nível mais alto de privilégio;

A taxonomia necessita de um modo de descrição de cada nível de privilégio, e um modo de descrever transições, e um modo de categorizar as ações. Essas três exigências são apresentadas nas seções seguintes.

4.1.3.1 Níveis de Privilégios

A taxonomia define uma aproximação para definir ranking dos níveis de privilégio. As categorias de privilégio que são aplicadas neste trabalho são:

R - acesso de rede Remoto

L - acesso de rede Local

U - acesso de Usuário

S - Root/ Acesso de Super Usuário

P - Acesso Físico ao Host

Ter privilégio “em nível” de acesso de rede Remoto, refere-se a ter, via um rede interligada de sistemas, um acesso mínimo a um sistema alvo. “Acesso de rede local” representa a capacidade de ler e escrever na rede local que o sistema alvo utiliza. “Acesso de Usuário” refere-se à capacidade de executar comandos de usuário normais em um sistema. “Acesso *Root*/ *Super-User*” dá ao usuário de um sistema o controle total do software. “O acesso Físico ao *host*” permite que o operador manipule fisicamente características do sistema (isto é retirar HDs, inserir disquetes, desligar o sistema). Esta lista só representa um subconjunto de todos os níveis de acesso possíveis, mas essas foram as categorias mais úteis para descrever os ataques na avaliação de detecção de intrusão do DARPA 1998.

4.1.3.2 Métodos de Transição ou Exploração

Um atacante tem de explorar um ponto fraco de um *“framework”* de segurança para executar um ataque. Os cinco métodos de transição que foram explorados para a avaliação do DARPA 1998, com as letras (**m**, **a**, **b**, **c**, **s**) usadas para representar os métodos, foram:

m) *Masquerading*: em alguns casos é possível enganar um sistema utilizando uma falsa representação. Os exemplos de *masquerading* incluem a utilização de *username/password* roubado ou envio de um pacote TCP com um endereço de fonte forjado.

a) Abuso de Característica: há ações legítimas que cada um pode executar, ou é esperado executar, e quando tomado ao extremo pode levar a falha do sistema. O exemplo inclui encher uma partição de disco com arquivos de usuário ou iniciar centenas de conexões de telnet a um host para encher a sua tabela de processo.

b) Implementação de Bug: um bug em um programa de confiança pode permitir que um ataque prossiga. Os exemplos específicos incluem estouros de buffers e condições de corrida.

c) Sistema Misconfiguration: um atacante pode explorar erros na configuração da política de segurança, que permite que o atacante opere em um nível mais alto de privilégio desejado.

s) Engenharia Social: um atacante pode ser capaz de coagir um operador humano do sistema.

Um ataque individual pode usar mais de um desses métodos. Por exemplo, um bug na implementação da pilha TCP/IP em alguns sistemas permite travar o sistema enviando um pacote TCP corrompido, cuidadosamente construído. Este pacote também pode ter o endereço de fonte forjado para se evitar a identificação do atacante. Tal ataque estaria explorando tanto masquerading, como uma implementação tipo bug, e seria possível descobrir a intrusão observando qualquer uma dessas características.

4.1.4 Ataques de Negação de Serviço (Denial of Service Attacks)

Um ataque de negação de serviço ocorre quando o atacante torna algum recurso do sistema ou uso da memória demasiadamente ocupado, impedindo o sistema de tratar pedidos legítimos; negando acessos legítimos aos usuários de uma máquina. Há muitas variedades de ataques de negação de serviço (ou DoS). Alguns ataques de DOS (como um *mailbomb*, Netuno, ou *ataque de smurf*) abusam de uma característica perfeitamente legítima. Os outros (*teardrop, Ping of Death*)

criam pacotes corrompidos que confundi a pilha TCP/IP da máquina que está tentando reconstruir o pacote. Há ainda outros (*apache2, back, syslogd*), que tiram proveito de *bugs* de um determinado *daemon* de rede. A figura 6-1 fornece um resumo dos ataques de negação de serviço usados na avaliação, DARPA 1998, de detecção de intrusão. Cada linha representa um único tipo do ataque. As seis colunas mostram o nome do ataque, uma lista dos serviços que os ataques exploram, as plataformas que são vulneráveis ao ataque, o tipo de mecanismo que é explorado pelo ataque (implementação de *bug*, abuso de característica, *masquerading*, ou *misconfiguration*), o tempo usado para implementar o ataque, e um resumo do efeito do ataque.

Tabela -1 Sumário de Ataques de Negação de Serviço

Nome	Serviço	Plataforma Vulnerável	Mecanismo	Tempo de Implementação	Efeito
Apache2	http	Apache	Abuso	Curto	Crash http
Back	http	Apache	Abuso/Bug	Curto	Resposta do servidor lenta
Land	N/D	SunOS	Bug	Curto	Pára a máquina
Mailbomb	smtp	Todas	Abuso	Curto	Annoyance
SYN Flood	TCP	Todas	Abuso	Curto	Nega serviço em uma ou mais portas por minutos
Pingo of Death	ICMP	Nenhum	Bug	Curto	Nenhum
Process Table	TCP	Todas	Abuso	Moderado	Nega Novos Processos
Smurf	ICMP	Todas	Abuso	Longo	Rede lenta
Syslogd	syslog	Solaris	Bug	Curto	Mata Syslog
Teardrop	N/A	Linux	Bug	Curto	Reboot
Udpstorm	Echo/charge	Todos	Abuso	Curto	Rede Lenta

4.1.5 Ataques User to Root

Exploits User-to-Root são uma classe de *exploits* na qual o atacante começa com acesso a uma conta de usuário normal no sistema (possivelmente através de um sniffer, um dicionário de ataque, ou engenharia social); e é capaz de explorar alguma vulnerabilidade para ganhar o acesso de *root* do sistema.

Há vários tipos diferentes de ataques *User-to-Roots*. O mais comum é o ataque de *buffer overflow*. A sobrecarga de *buffer* (*buffer overflow*) ocorre quando um programa copia dados demasiados em um buffer sem se assegurar que os

dados se ajustarão. Por exemplo, se um programa espera que o usuário introduza o nome de usuário, o programador deve decidir quantos caracteres o buffer do nome necessitará. Assumindo que o programa aloca 20 caracteres do buffer para nome, é que o nome de usuário tem 35 caracteres, 15 caracteres inundarão o buffer para nome. Quando este excesso ocorre, os 15 últimos caracteres são colocados na pilha, sobre escrevendo o conjunto de instruções a ser executado. Pela manipulação cuidadosa dos dados que transbordam na pilha, um o atacante pode fazer com que ordens arbitrarias sejam executadas pelo sistema operacional. Apesar do fato de os programadores eliminarem este problema com técnicas de programação cuidadosa, alguns utilitários comuns são susceptíveis a ataques de sobrecarga de buffer [27].

Tabela 2 - Sumário de User to Root

Nome	**Serviço**	**Plataforma Vulnerável**	**Mecanismo**	**Tempo de Implementação**	**Efeito**
Eject	Qualquer sessão de Usuário	Solaris	Estouro de Buffer	Médio	Shell do Root
Ffbconfig	Qualquer sessão de Usuário	Solaris	Estouro de Buffer	Médio	Shell do Root
Fdformat	Qualquer sessão de Usuário	Solaris	Estouro de Buffer	Médio	Shell do Root
Loadmodule	Qualquer sessão de Usuário	SunOS	Poor Environment Sanitation	Curto	Shell do Root
Perl	Qualquer sessão de Usuário	Linux	Poor Environment Sanitation	Curto	Shell do Root
Os	Qualquer sessão de Usuário	Solaris	Gerenciamneto de Temp Pobre	Curto	Shell do Root
Xterm	Qualquer sessão de Usuário	Linux	Estouro de Buffer	Curto	Shell do Root

Outra classe de ataques User-to-Roots explora programas que fazem suposições sobre o ambiente no qual eles estão sendo executados. Um bom exemplo deste ataque é o *loadmodule*. Há também ataque tipo *User-to-Roots* que tira proveito de programas que não têm cuidado sobre o modo como utilizam os arquivos temporários. Finalmente, algumas vulnerabilidades *User-to-Roots* existem como causa de ações conjuntas de um programa único, ou de dois ou mais programas rodando simultaneamente [26]. Embora a programação cuidadosa possa

eliminar todas essas vulnerabilidade, bugs como esses estão presentes em cada versão principal do UNIX e Windows da Microsoft disponíveis hoje. A tabela 2 resume os ataques User-to-Roots usados na avaliação do DARPA 1998. Observe que todos os ataques User-to-Roots podem ser dirigido por alguma sessão interativa de usuário (no console, ou interagindo por telnet ou login remoto), e todos estes ataques criam uma nova *shell* com privilégios de root.

4.1.6 Ataques Remote to User

Um ataque Remote to User ocorre quando um atacante tem a capacidade de enviar pacotes a uma máquina através da rede - mas não tem uma conta nesta máquina – e explora alguma vulnerabilidade para ganhar acesso local como usuário daquela máquina. Há muitos modos possíveis que um atacante pode obter acesso não autorizado a uma conta local em uma máquina.

Alguns ataques discutidos nesta categoria exploram excessos de buffers no software de servidor de rede (imap, named, sendmail). Os ataques Dictionary, Ftp-Write, Guest e Xsnoop , tentam explorar a política fraca de segurança do sistema ou a má configuração. O ataque Xlock envolve engenharia social - para o ataque ter sucesso o atacante deve imitar com sucesso um operador humano no fornecimento de sua senha a um screensaver, que é de fato um cavalo de tróia. A tabela 4-3 resume as características deste tipo de ataque, o que esteve incluído na avaliação de detenção de intrusão do DARPA 1998.

4.1.7 Probes

Nos últimos anos, um número crescente de programas tem sido distribuído, os quais podem vasculhar automaticamente uma rede de computadores e reunir informação ou achar alguma vulnerabilidade conhecida [28]. Esses probes de rede são bastante úteis a um atacante que está organizando um futuro ataque. Um atacante com um mapa de qual máquina e serviços estão disponíveis na rede, pode usar estas informação para procurar pontos fracos. Alguns desses instrumentos de exploração (satan, saint, mscan) permitem que até um atacante muito inexperiente, rapidamente verifique centenas ou milhares de máquinas em uma rede atrás de uma

vulnerabilidade conhecida. A tabela 4 fornece um sumário dos probes que são apresentados nesta dissertação.

Tabela -3 Sumário de Remote to User

Nome	Serviço	Plataforma Vulnerável	Mecanismo	Tempo de Implementação	Efeito
Dictionary	telnet, rlogin, pop, imap, ftp	Todas	Abuso de Característica	Médio	Acesso a User-Level
Ftp-write	ftp	Todas	Misconfiguration	Curto	Acesso a User-Level
Guest	telnet, rlogin	Todas	Misconfiguration	Curto	Acesso a User-Level
Imap	imap	Linux	Bug	Curto	Shell do Root
Named	dns	Linux	Bug	Curto	Shell do Root
Phf	http	Todos	Bug	Curto	Executa comando como usuário http
Sendmail	smtp	Linux	Bug	Longo	Executa comando como root
Xlock	-	Todos	Misconfiguration	Médio	Spoof usuário para obter senha
Xsnoop	-	Todos	Misconfiguration	Curto	Monitora Digitação remotamente

4.2 Dados de Entrada

Para se poder realizar os testes em redes neurais como classificadoras de IDS, uma base de dados com padrões de ataques e padrões de tráfego normais foi necessária. Como se pode ver em [19, 27-31] tem-se disponível uma base de dados que tem sido utilizada há muito tempo e que foi utilizada no KDD CUP 99. Esta base de dados é oriunda do primeiro esforço para testes de sistemas de detecção de intrusão, que foi feito pelo DARPA em 1998 e 1999. Alguns pesquisadores ainda se utilizam da base de dados produzida pelo DARPA, como se pode observar em [32-40]. Estas são base de dados prontas que se pode utilizar para os teste de avaliação de desempenho em IDS´s, porém, têm-se casos em que o pesquisador optou por desenvolver a sua própria base de dados em [41-47]

A base de dados disponibilizada pelo KDD CUP99 está descrita na Tabela 7, onde se pode ver o número de dados ou linhas contidos em cada uma das bases, bem como a quantidade de ataques contidos em cada um dos conjuntos. Para

separar o *dataset* em 5 conjuntos distintos, cada um representando uma classe de tráfego, criou-se um programa com a finalidade de criar 5 arquivos. Cada arquivo contém um tipo específico de tráfego, ou seja, um arquivo para tráfego normal, um para Remote-to-Local, outro para User-to-Root, outro para Probe e outro para DoS. Com isso, foi possível levantar a quantidade de padrões contidos em cada uma das classes para criar os conjuntos de treinamento, teste e validação, com o mesmo número de padrões de ataque e tráfego normal.

Tabela -4 Sumário de Probes

Nome	**Serviço**	**Plataforma Vulnerável**	**Mecanismo**	**Tempo de Implementação**	**Efeito**
Ipsweep	ICMP	Todas	Abuso de Característica	Curto	Encontra máquinas Ativas
Mscsn	Muitos	Todas	Abuso de Característica	Curto	Procura por vulnerabilidares conhecidas
Nmap	Muitos	Todas	Abuso de Característica	Curto	Encontra portas ativas em uma máquina
Saint	Muitos	Todas	Abuso de Característica	Curto	Procura por vulnerabilidares conhecidas
Satan	Muitos	Todas	Abuso de Característica	Curto	Procura por vulnerabilidares conhecidas

Pode-se ver o resultado desta contagem nas Tabelas 5 e 6, onde pode nota que no conjunto de validação, a presença de padrões de ataques desconhecidos pela rede neural. Para alguns padrões de ataque, utilizou-se toda a base disponível, visto que a quantidade era pequena, para as outras classes, onde o número de padrões é alto, fez-se uma amostragem aleatória da base, buscando um número suficiente para o treinamento e que contemplasse todos os tipos de ataques.

Como descrito anteriormente, optou-se por utilizar uma rede neural por tipo de ataque, as quais foram treinadas primeiramente com os ataques descritos na tabela 5, e depois utilizou-se os ataques descritos na tabela 6 verificando se as redes neurais seriam capazes de identificar novos ataques contidos neste conjunto de dados.

Tabela -5 Classificação dos Ataques para o Training e Testing Sets

U2R	R2L	Probe	DoS
loadmodule	spy	portsweep	back
buffer_overflow	ftp_write	ipsweep	land
perl	guess_passwd	satan	neptune
rootkit	imap	nmap	pod
	phf		smurf
	multihop		teardrop
	warezclient		
	warezmaster		

Tabela -6 Classificação dos Ataques para o Validation Set

U2R	R2L	Probe	DoS
xterm	worm	ipsweep	Smurf
sqlattack	xlock	mscan	teardrop
Os	xsnoop	nmap	udpstorm
rootkit	snmpgetattack	portsweep	warezmaster
perl	snmpguess	saint	processtable
loadmodule	sendmail	satan	Pod
httptunnel	phf		neptune
buffer_overflow	multihop		Land
	imap		mailbomb
	ftp_write		apache2
	guess_passwd		Back
	named		

4.3 Tratamento dos Dados

Cada linha do *"dataset"* é composta por 41 entradas descritas na tabela 7, onde tem-se os dados discretos e contínuos; porém para treinamento usou-se dados contínuos e normalizados. Sendo assim, os valores máximos e mínimos para os valores contínuos a serem normalizados e os valores discretos, associou-se valores entre 0,0 e 1,0. Fez-se então um levantamento dos valores de máximo e mínimo para cada campo, a fim de normalizar os dados para o treinamento das redes neurais. Porém no campo número 3, obteve-se 63 valores diferentes, o que dá 1/63, ou seja, cada valor discreto corresponde a aproximadamente 0,016, que é um valor muito pequeno e não apresentaria diferença significativa entre um campo e outro para a rede neural.

Para esse campo, optou-se por uma representação binária, utilizando 7(sete) bits para percorrer os 63 valores disponíveis no campo e assim passou-se de 41 entradas para 46 entradas. Para os valores contínuos foi utilizado 10 degraus, quando a diferença entre o mínimo e o máximo era menor que 512, e 20 degraus para os campos 05 e 06; quando a diferença era menor que 20 usou-se o valor máximo para normalizar os demais valores.

Tabela -7 Número de Padrões no Training Set e Validation Set

Dados do Training Set		Dados do Validation Set	
Total de linhas	494022	Total de linhas	311030
Total de ataques	396743	Total de ataques	250436
Total de dos	391458	Total de dos	231455
Total de u2r	52	Total de u2r	246
Total de r2l	1126	Total de r2l	14569
Total de probe	4107	Total de probe	4166
Total normal	97277	Total normal	60593

Tabela -8 Descrição dos Campos de Entrada

Nº	Campo	tipo	degraus
01	Duration	continuous	10
02	protocol_type	symbolic.	3
03	Service	symbolic	67 (binário)
04	flag:	symbolic	11
05	src_bytes:	continuous	20
06	dst_bytes	continuous	20
07	Land	symbolic	10
08	wrong_fragment	continuous	10
09	Urgent	continuous	10
10	Hot	continuous	10
11	num_failed_logins	continuous	10
12	logged_in	symbolic.	10
13	num_compromised	continuous	10
14	root_shell	continuous	10
15	su_attempted	continuous	10
16	num_root	continuous	10
17	num_file_creations	continuous	10
18	num_shells	continuous	10
19	num_access_files	continuous	10
20	num_outbound_cmds	continuous	10
21	is_host_login	symbolic	10
22	is_guest_login	symbolic	10
23	Count	continuous	10
24	srv_count	continuous	10
25	serror_rate:	continuous	10
26	srv_serror_rate	continuous	10
27	rerror_rate	continuous	10
28	srv_rerror_rate	continuous.	10
29	same_srv_rate	continuous	10
30	diff_srv_rate	continuous	10
31	srv_diff_host_rate	continuous	10
32	dst_host_count	continuous	10
33	dst_host_srv_count	continuous	10
34	dst_host_same_srv_rate	continuous	10
35	dst_host_diff_srv_rate	continuous	10
36	dst_host_same_src_port_rate	continuous	10
37	dst_host_srv_diff_host_rate	continuous	10
38	dst_host_serror_rate	continuous	10
39	dst_host_srv_serror_rate	continuous	10
40	dst_host_rerror_rate	continuous	10
41	dst_host_srv_rerror_rate	continuous	10

4.4 Treinamento da Rede MLP

A MLP possui quarenta e seis unidades de entrada descritas na tabela 9 e duas unidades de saída. As unidades de saída têm funções lineares de ativação, enquanto que as unidades “hidden” possuem funções sigmoidais de ativação. Várias arquiteturas foram testadas, incluindo de 5 a 25 unidades hiddens.

Tabela -9 Dados dos Campos empregados na Representação

Unidade	Dado do Campo	Unidade	Dado do Campo
1	duration	27	is guest login
2-8	service	28	count
9	ag	29	srv count
10	src bytes	30	serror rate
11	dst bytes	31	srv serror rate
12	land	32	rerror rate
13	wrong fragment	33	srv rerror rate
14	urgent	34	same srv rate
15	hot	35	di_ srv rate
16	num failed logins	36	srv di_ host rate
17	logged in	37	dst host count
18	num compromised	38	dst host srv count
19	root shell	39	dst host same srv rate
20	su attempted	40	dst host di_ srv rate
21	num root	41	dst host same src port rate
22	num _le creations	42	dst host srv di_ host rate
23	num shells	43	dst host serror rate
24	num access _les	44	dst host srv serror rate
25	num outbound cmds	45	dst host rerror rate
26	is host login	46	dst host srv rerror rate

O treinamento usado para inserir conhecimento nas redes neurais foi executado utilizando o cross validation. Primeiramente, dividiu-se o *dataset* de treinamento em duas partes, uma com 70% de padrões e outra com 30%. Definiu-se a maior parte como sendo o training set, e a menor como sendo o validation set. Para o treinamento também se desenvolveu dois programas, um para o treinamento da rede, e o outro para a validação do treinamento. Deste modo, treinou-se a rede utilizando o *training set* com o programa de treinamento, e depois verificou-se o erro com o *testing set*, utilizando o programa de validação. Para iniciar, criou-se os pesos aleatoriamente, definindo uma semente para a geração dos números aleatórios e definindo o valor mínimo e máximo para os pesos. A learning rate é reduzida em

50% quando o erro total aumenta, e aumentado em 2% quando o erro diminui. Momentum é desabilitado até o fim do treinamento, caso o erro aumente.

Feito isso, executou-se a primeira bateria de treinamento, que consistiu em treinar a rede para uma certa quantidade de épocas, gerando novos pesos baseados nos pesos iniciais e nos padrões do training set, como pode-se ver na figura 4-3. Verificando o erro com o programa de validação para o conjunto de pesos gerados pelo programa de treinamento. Se o erro encontrado era menor que o anterior, executou-se outra bateria de treinamento, partindo com o conjunto de pesos validados como valores iniciais e assim reduzir o erro. Se o erro aumentasse, reduziu-se a quantidade de épocas de treinamento e verificou-se o resultado, caso fosse melhor, validava-se o conjunto de pesos e repetia o processo para o mesmo número de épocas. Caso o conjunto fosse pior, reduzia-se o número de épocas novamente e repetia o processo até o número de épocas chegarem a zero.

Terminada a validação, ou seja, para qualquer número de épocas utilizadas no treinamento o erro aumenta, partiu-se para a etapa de verificação e teste da rede neural. Nesta etapa, utilizaram-se os pesos obtidos na etapa anterior e um conjunto de padrões desconhecidos pela rede, com novos tipos de ataques e padrões normais para testar a capacidade de generalização e a eficiência no reconhecimento de padrões.

Através do programa de validação e da utilização de um *dataset* composto de padrões de ataques desconhecidos, bem como padrões normais*(validation set)*, verificou-se o erro global, e quais os padrões que não foram bem classificados e assim obteve-se os resultados.

4.4.1 Ataques User-to-Root

O primeiro experimento pretendeu detectar ataques do tipo user-to-root. Utilizou-se um training set com 37 padrões normais e de ataque, sendo 17 ataques do tipo buffer-overflow, 9 tipo loadmodule, 3 tipo perl , e 8 tipo rootkit. O testing set com 37 padrões normais e de ataque, sendo 17 ataques do tipo buffer-overflow, 9 do tipo loadmodule, 3 do tipo perl e 8 do tipo rootkit, como no training set. O validation set continha 228 padrões normais e de ataque, sendo 13 attacks do tipo xterm, 2 do

tipo sqlattack , 16 do tipo ps, 13 do tipo rootkit, 2 do tipo perl , 2 do tipo loadmodule, 22 do tipo buffer overflow, e 158 do tipo httptunnel.

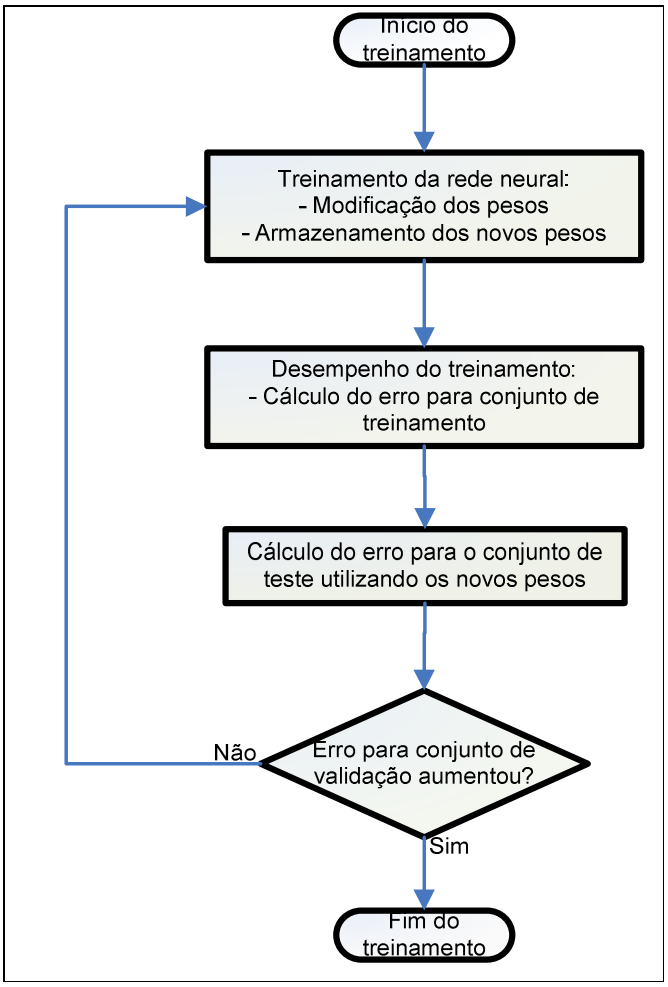

Figura 4-3 Processo de Treinamento

A topologia empregada no experimento foi de 46 unidades de entrada, 5 unidades hiddens e 2 unidades de saídas. Para obter o melhor resultado no treinamento foram necessárias 6400 épocas de treinamento.

4.4.2 Ataques Remote-to-Local

O segundo experimento pretendeu detectar ataques do tipo remote-to-local. Utilizou-se um training set com 328 padrões normais e de ataque, sendo 2 ataques do tipo spy, 8 de ftp write, 44 do tipo guess passwd, 35 do tipo imap, 5 do tipo phf , 6

do tipo multihop, 144 do tipo warezclient, e 84 do tipowarezmaster. O testing set cotinha 78 padrões normais de ataque, sendo 2 do tipo spy, 4 do tipo ftp write, 4 do tipo guess passwd, 2 do tipo phf , 3 do tipo multihop, 61 do tipo warezclient, e 2 do tipo warezmaster. O validation set continha 336 padrões normais de ataque, sendo 1 ataque do tipo worm, 9 do tipo xlock, 4 do tipo xsnoop, 219 do tipo snmpgetattack , 19 do tipo snmpguess, 11 do tipo sendmail, 1 do tipo phf , 3 do tipo ftp write, 59 do tipo guess passwd e 10 do tipo named.

A topologia empregada para este experimento foi de 46 unidades de entradas, 20 unidades hiddens e 2 unidades de saídas. Para obter o melhor resultado no treinamento foram necessárias 3000 épocas de treinamento.

4.4.3 Ataques Probe

O terceiro experimento pretendeu detectar ataques do tipo probe. O training set continha 688 padrões normais e 688 padrões de ataque, sendo 137 ataques do tipo portsweep, 134 do tipo ipsweep, 360 do tipo satan, e 57 do tipo nmap. O testing set continha 387 padrões normais e 387 padrões de ataque, sendo 137 do tipo portsweep, 158 do tipo ipsweep, e 92 do tipo satan. O validation set continha 1284 padrões normais e 1284 padrões de ataque, sendo 51 ataques do tipo ipsweep, 529 do tipo mscan, 35 do tipo nmap, 41 do tipo portsweep, 206 do tipo saint e 422 do tipo satan.

A topologia empregada para este experimento foi de 46 unidades de entradas, 20 unidades hiddens e 2 unidades de saídas. Para obter o melhor resultado no treinamento foram necessárias 2000 épocas de treinamento.

4.4.4 Ataque de Negação de Serviço (DoS)

O quarto experimento teve como objetivo detectar ataques do tipo “denial-of-service”. O training set continha 613 padrões normais e 613 padrões de ataque, sendo 148 ataques do tipo back, 56 do tipo land, 253 do tipo neptune, 86 do tipo pod, 34 do tipo smurf e 36 do tipo teardrop. O testing set continha 207 padrões normais e 207 padrões de ataque, sendo 50 do tipo back, 19 do tipo land, 85 do tipo neptune, 29 do tipo pod, 12 do tipo smurf , e 12 do tipo teardrop. O validation set

continha 570 padrões normais e 570 padrões de ataque, sendo 60 atauqes do tipo back, 9 do tipo land, 111 do tipo neptune, 23 do tipo pod, 55 do tipo smurf , 12 do tipo teardrop, 2 do tipo udpstorm, 151 do tipo processtable, 97 do tipo mailbomb e 50 do tipo apache2.

A topologia empregada para este experimento foi de 46 unidades de entradas, 16 unidades hiddens e 2 unidades de saídas. Para obter o melhor resultado no treinamento foram necessárias 3700 épocas de treinamento.

5 RESULTADOS

O principal objetivo deste experimento é obter de um Sistema de Detecção de Intrusão, a capacidade de detectar ataques novos ou desconhecidos. Porém, vale ressaltar, que para obter um bom resultado, ele precisa ter uma baixa taxa de falsos positivos e falsos negativos. Se um IDS tem uma ótima taxa de detecção de ataques, inclusive ataques novos, porém com uma taxa relativamente alta de falsos positivos, haverá problemas, pois os tráfegos normais estão sendo classificados com ataques. Sendo assim, o administrador terá alarmes para a maioria do tráfego que passa pela sua rede, sendo uma boa parte falsos alarmes, provenientes do tráfego normal. Por outro lado, se um IDS tem baixa taxa de falso positivo e alta taxa de falso negativo, a rede em que o sistema monitora estará vulnerável e o sistema não informará a maioria dos ataques sofridos.

Deste modo, deseja-se uma ferramenta que tenha a capacidade de detecção de novos ataques e que possua uma taxa baixa de falsos positivos, para que esta se torne confiável. Com isso, a partir dos treinamentos descritos no capítulo anterior, obtiveram-se os seguintes resultados para cada um dos experimentos.

5.1 User-to-Root

Para o primeiro experimento, que tinha como objetivo detectar ataques do tipo User-To-Root, obteve-se 100% de detecção e 0% de falso alarme , ou seja, todos os padrões contidos no validation set foram classificados corretamente.

Como podemos ver, para este tipo de ataque, conseguiu-se o ideal, ou seja, nenhum falso negativo e nenhum falso positivo. A rede neural foi capaz de identificar os novos ataques, bem como diferenciá-los do tráfego normal, atingindo um excelente resultado.

Tabela 10 - Matriz Confusão User-to-Root

	User-To-Root	Normal
User-To-Root	228	0
Normal	0	228
%	100	100

5.2 Remote-to-Local

No segundo experimento, voltado para ataques Remote-To-Local, obtivemos 11 falsos negativos e 23 falsos positivos, ou seja, 11 ataques foram classificados como tráfego normal e 23 padrões de tráfego normal foram classificados como ataques. Dentre os 11 ataques classificados como normal encontrou-se os 4 ataques tipo xock, 1 tipo xsnoop, 1 tipo sendmail, 1 tipo phf, 1 tipo ftp_write e 3 tipo named.

Para esse tipo de ataque, a rede neural teve um bom desempenho, com uma taxa de detecção acima de 95% e obtendo apenas 6,15% de falso positivo. Deste modo ela poderia ser utilizada na detecção de intrusão, pois mesmo quando aparecer um falso positivo, devido a sua baixa incidência, não irá atrapalhar o trabalho de triagem de alarmes do administrador da rede.

Tabela 11 - Matriz Confusão Remote-to-Local

	Remote-to-Local	Normal
Remote-to-Local	325	23
Normal	11	313
%	96,73	93,15

5.3 Probe

No terceiro experimento, voltado para ataques de Probe, obteve-se 100% na detecção de ataques, porém 61 padrões normais foram classificados como ataques ou falsos positivos.

Para este experimento, a rede teve um excelente desempenho na detecção dos ataques, porém a taxa de falsos positivos foi a maior entre todas as redes, ou seja 7,75%. Mesmo assim poderia-se utilizar a rede neural para detecção, já que todos os ataques foram detectados e apenas alguns padrões de tráfego normal foram classificados como ataque, o que acarretaria em um pouco mais de trabalho para o administrador.

Tabela 12 - Matriz Confusão Probe

	Probe	Normal
Probe	1294	61
Normal	0	1223
%	100%	92,25

5.4 Negação de Serviço (DoS)

No quarto e último experimento, voltado para ataques de DoS, obteve-se 17 falsos negativos e 26 falsos positivos. Dentre os 17 ataques classificados como

tráfego normal, 1 ataque do tipo pod, 2 do tipo teardrop, 2 do tipo udpstorm e 12 do tipo processtable.

Para ataque do tipo DoS, a rede teve uma boa taxa de detecção, acima de 97%, e uma taxa de falso positivo aceitável, 5,56%. Com essas taxas, também seria possível a utilização desta rede na detecção de intrusão, o que ajudaria no combate aos ataques de negação de serviço e mesmo os falsos positivos, que são de pequena grandeza, não iriam ocupar muito tempo do administrador na verificação da veracidade dos alarmes gerados.

Tabela 13 - Matriz Confusão DoS

	DoS	Normal
Dos	553	26
Normal	17	544
%	97,02	95.44

5.5 Comparação dos Resultados

Como dito anteriormente, está se buscando resultados bons na detecção de ataques novos e que o sistema não gere muitos falsos positivos. Como se pode ver conseguiu-se chegar bem perto da condição ideal, que seria 100% de detecção e 0% de falso positivo, em quase todos os experimentos.

Para poder avaliar melhor os resultados, segue abaixo uma comparação com resultados obtidos utilizando outros tipos de tecnologia e que utilizaram a mesma base de dados. Utilizou-se os resultados obtidos utilizando-se Supporting Vector Machines (SVM)[48], Regression Tree (R.T.) [30], Self Organized Maps (SOM)[31] e Multivariate Adaptative Regression Splines (MARS)[36]. O resultado desta comparação pode ser visto na tabela 14 e na figura 5.1.

Tabela 14 Comparação de Desempenho

	MLP	SOM 2 layer	SVM	MARS	R.T.
Normal	95,96	92,40	99,87	96,08	99,64
DoS	97,02	96,50	99,75	94,73	99,47
Probe	100,00	72,80	99,70	92,32	97,85
U2R	100,00	22,90	99,87	99,71	48,00
R2L	96,73	11,30	99,83	99,48	90,58

Como se pode ver, o modelo utilizado neste experimento (MLP), teve um desempenho muito próximo comparado aos que se utilizaram de outros tipos de implementações na detecção de ataques e em dois casos (Probe e U2R) teve o melhor desempenho entre todos, atingindo 100%.

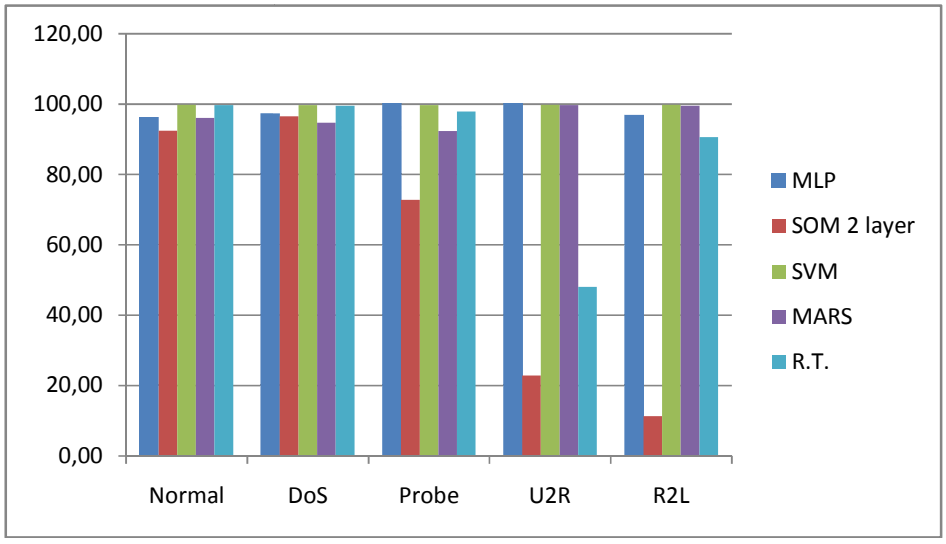

Figura 5-1 Gráfico Comparativo de Desempenho

6 CONCLUSÕES

Os IDS´s *ruled-based* ou *signature-based* necessitam que suas bases de dados estejam sempre atualizadas, contendo informações sobre os tipos de ataques, para que estes possam detectar e informar ao administrador sobre as ocorrências. Uma vez que o administrador confia neste sistema para a detecção de ataques, é imprescindível que ele detecte qualquer tentativa de invasão na rede. Caso isto não ocorra, o IDS se torna uma brecha de segurança, por não estar com a base de dados atualizada não será capaz de detectar uma invasão; e o administrador não ficará sabendo de tal tentativa, e quando souber será tarde para tomar medidas ostensivas, nesses casos medidas corretivas e restauração do sistema.

Para não ficar totalmente dependente das atualizações da base de dados, foi proposto o uso de redes neurais artificiais na detecção de intrusão. Com o uso de redes neurais, a partir de sua capacidade de generalização no quesito reconhecimento de padrão, os IDS´s passaram a detectar novos ataques, que previamente não apareciam na sua base de dados; ou seja, detecção de ataques desconhecidos através do reconhecimento de padrões semelhantes de ataques.

Pôde-se observar que a utilização de redes neurais MLP para a detecção de intrusão[49] trouxe resultados satisfatórios, pois se conseguiu detectar tantos os ataques conhecidos, como também os novos ataques e, portanto desconhecidos.

Este era o principal objetivo deste trabalho. Nota-se também que houve um baixo nível de falsos negativos, o que favorece também ao administrador, pois não é de muita valia um sistema detectar novos ataques, porém com um número de falsos positivos elevado, o que tomaria muito tempo e atenção em um falso alerta.

Dentre os resultados obtidos, a rede que tinha como finalidade detectar ataque do tipo U2R obteve o melhor desempenho, pois detectou todos os ataques e tráfegos normais corretamente. Outra rede que obteve 100% de acerto foi a do tipo Probe, que detectou todos os ataques, e teve uma taxa de 95,25% de acerto para o tráfego normal. Tanto a rede para o tipo R2L como a do tipo DoS obtiveram altas taxas de acertos, todas acima de 90%, sendo 93,15% para tráfego normal e 96,73% para ataques tipo R2L e 95,44% para tráfego normal e 97,02% para ataques. [49]

Com isso a meta foi atingida e obteve-se bons resultados com este tipo de rede neural artificial.

6.1 Trabalhos Futuros

Propõe-se como trabalhos futuros, estudos com outros tipos de redes neurais para uma avaliação de desempenho das mesmas, a fim de verificar qual modelo possa se adequar melhor no reconhecimento de padrão, e com melhor tempo de treinamento e de resposta. A utilização de topologias hierárquicas também pode ser uma proposta futura, a utilização de lógica fuzzy na tomada de decisões em modelos híbridos, que utilizem sistemas *ruled-based com* juntamente com as redes neurais, algoritmos genéticos ou conjuntos aproximados (rough sets) para a redução das informações de entrada e treinamento mais rápido e eficaz. Também pode-se citar o desenvolvimento de uma ferramenta a partir deste trabalho, não mais apenas um experimento, mas um sistema de detecção de intrusão que possa ser instalado em uma empresa; ou ainda o desenvolvimento de melhorias em ferramentas de código-livre, como o snort.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] P. Miller, A. Inoue, “Colaborative Intrusion Detection System”, Departament of Computer Science, Eastern Washington University, 2003.

[2] D. Denning, “An Intrusion Detection System,” *Proc. Symp. Security and Privacy*, IEEE Computer Soc. Press, Los Alamitos, Calif., 1986, pp. 118–131.

[3] Cisco Systems, Inc. “NetRanger Intrusion Detection System Technical Overview,” http://www.cisco.com/univercd/cc/td/doc/product/iaabu/netrangr/index.htm.

[4] Lawrence Livermore National Laboratory. “Network Intrusion Detector (NID) Overview,” Computer Security Technology Center, http://ciac.llnl.gov/cstc/nid/.

[5] V. Paxson, “Bro: A System for Detecting Network Intruders in Real-Time,” *Proc. Seventh Usenix Security Symp.*, Usenix Assoc., Berkeley, Calif., 1998.

[6] Kemmerer, Rachard A. “NSTAT: A Model-based Real-time Network Intrusion Detection System,” Computer Science Department, University of California, Santa Barbara, Report TRCS97-18, http://www.cs.ucsb.edu/TRs/TRCS97-18.html.

[7] D. Joo, T. Hong, I. Han, “The Neural Network Models for IDS Based on The Asymmetric Cost of False Negative Errors and False Positive Erros”, Expert System with Applications 25, Elsevier Science Ltd. 2003.

[8] C. Zhang, J. Jiang, M. Kamel, “Intrusion detection using hierarchical neural networks”, International Association for Pattern Recognition, Elsevier Science Inc., 2005.

[9] A. R. Baker, B. Caswell, M. Poor, “Snort 2.1 Intrusion Detection Second Edition”, Syngress Publishing, 2004.

[10] S. Haykim, “Neural Networks a Comprehensive Foundation. Second Edition”, Prentice Hall, 1999.

[11] W. S. McCulloch, W. Pitts, “A Logical Calculus of the Ideas Immanent in Nervous Activity”. Bulletin of Mathematical Biophysics, Vol.5, 1942.

[12] F. Rosenblatt, “The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain”. Psychological Review, vol 65, 1958.

[13] B. Widrow, M. E. J. Hoff, “Adaptive Switching Circuits”. IRE WESCON Convention Record, 1960.

[14] M. L. Minsky, S. A. Papert, “Perceptrons”. MIT Press, Cambridge, MA, 1969.

[15] D. E. Rummelhart, G. E. Hinton, J. R. Williams, “Internal Representation by Error Propagation”, in Parellel Distributed Processing. MIT Press, Cambridge, 1986.

[16] K. KNIGHT, “Connection Ideas and Algorithms. Communications of the ACM.”, November 1990/Vol.33,No.11.

[17] R. P. Lippmann, D. Fried, I. Graf, J. Haines, K. Kendall, D. McClung,S. Webber, S. Webster, D. Wyschograd, R. Cunninghan, M. Zissmam, ”Evaluating intrusion detection systems: The 1998 DARPA off-line intrusion detection evaluation”. In *Proceedings of the on DARPA Information Survivability Conference and Exposition* (DISCEX ’00, Hilton Head, South Carolina, Jan. 25-27). IEEE Computer Society Press, Los Alamitos, CA, 12–26.

[18] R. Lippmann, J. Haines, D. Fried, ”The 1999 DARPA off-line intrusion detection evaluation”. *Computer Networks*, 34, 2000.

[19] KDD CUP 99, http://kdd.ics.uci.edu//databases/kddcup99/kddcup99.html.

[20]K. Kendall, “A Database of Computer Attacks for the Evaluation of Intrusion Detection Systems”.*Master Thesis, MIT*, June 1999.

[21] Rootshell Website. http://ftp2.de.freebsd.org/pub/misc/www.rootshell.com/.

[22] Computer Emergency Response Team Website. http://www.cert.org.

[23] Internet Security Systems X-Force. http://xforce.iss.net/.

[24] D. Weber, “A Taxonomy of Computer Intrusions”, Master’s thesis, Massachusetts Institute of Technology, Cambridge, MA, 02139, 1998.

[25] Anonymous. Maximum Security: A Hacker’s Guide to Protecting Your Internet Site and Network, Chapter 15, pp. 359-362. Sams.net , 201 West 103rd Street, Indianapolis, IN, 46290. 1997.

[26] S. Garfinkel, G. Spafford, “Practical Unix & Internet Security”. O’Reilly & Associates,Inc., 101 Morris Street, Sebastopol CA, 95472, 2nd edition, April 1996.

[27] G. K. Kuchimanchi, V. V. Phoha, K. S. Balagni, S. R. Gaddam, “Dimension Reduction Using feature Extration Methods for Real-Time Misuse Detection System”, Workshop on Information Assurance and Security, 2004

[28] S. Chebrolu, A. Abraham, J. P. Thomas, ”Feature deduction and ensemble design of intrusion detection systems”, Computers Security, Volume 24, Issue 4 , pp. 295-307, 2005.

[29] H. G.Kayacik, A. N. Zincir-Heywood, M. I. Heywood, “On the capability of an som based intrusion detection system,” in Proceedings of the International Joint Conference on Neural Networks, vol. 3, pp. 1808–1813. IEEE, IEEE, July 2003.

[30] P.Laskov, P. Düssel, C. Schäfer, K. Rieck, ”Learning Intrusion Detection: supervised or unsupervised?”, Fraunhofer – FIRST.IDA.

[31] C. Zhang, J. Jiang, M Kamel, “Comparison of BPL and RBF Network in Intrusion Detection System”, Pattern Analysis and Machine Intelligence Research Group, University of Waterloo, Canadá.

[32] S. Mukkamala, G. Janoski, A. Sung, “Intrusion Detection Using Neural Networks and Support Vector Machines”, Department of Computer Science, New Mexico Institute of Mining and Tecnology, 2002.

[33] A. H. Sung, S. Mukkamala, “Identifying Important Features for Intrusion Detection Using Support Vector Machines and Neural Networks”, New Mexico Institute of Mining and Technology, 2003.

[34] S. Mukkamala, A. H. Sung, a. Abraham, “Intrusion Detection Using na Ensemble of Intelligent Paradigms”, Jornal of Network and Computer Application 28, 2005.

[35] A. K. Ghosh, A. Scwartzbard, M. Schatz, “Learning Program Behaivor Profiles for Intrusion Detection”, Proceeding of the Workshop on Intrusion Detection and Network Monitoring, Santa Clara, California, 1999.

[36] J. B. D. Cabrera, R. K. Mehra, “Control and Estimation Methods in Information Assurance – A Tutorial on Intrusion Detection Systems”, Conference on Decision and Control, Las Vegas, Nevada, 2002.

[37] A. K. Ghosh, A. Schwartzbard, “A Study in Using Neural Networks for Anomaly and Misuse Detection”, Proceedings of the 8th USENIX Security Symposium, Washington, D.C., 1999.

[38] R. P. Lippmann, R. K. Cunningham, “Improving Intrusion Detection performance Using Keyword Selection and Neural Netwworks”, Computer Networks 34, 2000.

[39] W. Lee, S. J. Stolfo, K. W. Mok, ”A Data Minig Framework for Building Intrusion Detection Models”, Computer Science Department, Columbia University.

[40] J. W. Haine, L. M. Rossey, R. P. Lippmann, R. K. Cunninghann, “Extendind the DARPA off-line Intrusion Detection Evaluations”, Lincon laboratory, Massachusetts Institute of Technology.

[41] E. H. Spafford, D. Zamboni, “Intrusion Detection Using Autonomous Agents”, Computer Networks 34, 2000.

[42] G. Vigna, W. Robertson, V. Kher, R A. Kemmerer, “A Stateful Intrusion Detection System for World-Wide Web Servers”, Reliable Software Group, Department of Computer Science, University of California, 2003.

[43] L. O. Hall, X. Liu, K. W. Bowyer, R. Banfield, “Why are Neural Networks Sometimes Much More Accurate than Decicsion Trees: An Analysis on a Bio-Informatics Problem”, Computer Science & Engineering, 2003.

[44] J. Li, G. Zhang, G. Gu, “The Research and Implemetation of Inteligent Intrusion Detection System Based on Artificial neural Network”, Proceedings of the Third International Conference on Machine Learning and Cybernetics, Shanghai, 2004.

[45] J. M. Bonifácio Júnior, A.M. Cansian, A. C. P. L. F. de Carvalho, E. S. Moreira, Instituto e Ciências Matemáticas de São Carlos – Universidade de São Paulo, 1998.

[46] S. Cho, “Incorporating Soft Computing Techniques Into a Probabilistcs Intrusion Detectiom System”, IEEE Transactions on System, Man, and Cybernetics, vol 32, 2002.

[47] S. C. Lee, D. V. Heinbuch, “Training a neural Network Based Intrusion Detector to recognize Novel Attacks”, IEEE Transactions on System, Man, and Cybernetics, vol 31, 2001.

[48] F. Liu, Z. Chen, “Intrusion Detection Based on Multi-Layer Minimax Probability Machine Classifier”, Proceedings of the Third International Conference on Machine Learning and Cybernetics, Shanghai, August 2004.

[49] O. A. S. Carpinteiro, R. S. Netto, I. Lima, A. C. Z. Souza, E. M. Moreira, “ A neural model in Intrusion Detection Systems”, Lecture Notes in Computer Science, Artificial Neural Networks – ICANN 2006.