UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

OTÁVIO BARCELOS GASPARETO

Redes Neurais Artificiais Aplicadas ao Reconhecimento de *Speed Cheating* em Jogos *Online* de Computador

Dissertação apresentada como requisito parcial
para a obtenção do grau de
Mestre em Ciência da Computação

Prof. Dr. Dante Augusto Couto Barone
Orientador

Porto Alegre, maio de 2008

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Gaspareto, Otávio Barcelos

Redes Neurais Artificiais Aplicadas ao Reconhecimento de *Speed Cheating* em Jogos *Online* de Computador / Otávio Barcelos Gaspareto. – Porto Alegre: PPGC da UFRGS, 2008.

59 f.: il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2008. Orientador: Dante Augusto Couto Barone.

1. Inteligência artificial. 2. Redes neurais artificiais. 3. Jogos de computador. 4. Trapaças. I. Barone, Dante Augusto Couto. II. Título.

“All men by nature desire knowledge.”
— ARISTOTLE

AGRADECIMENTOS

- Ao meu orientador, prof^{o} Dante Barone, pelo voto de confiança, amizade e por ter me guiado durante o decorrer do mestrado. Ao meu "co-orientador", Dr. André Schneider, pela amizade, paciência e trocas de idéias;
- Aos meus pais, Ivan e Elusa, por tudo o que representam em minha vida. Serei eternamente grato pelos ensinamentos, afeto e imagem que vocês me deram. Ao meu irmão Patrick, pelo entusiasmo em baixar a cabeça fazer pequisa;
- Aos amigos de longa data, em especial, Francisco Socal, Frederico Zerfass, Guilherme Lazzari, Eduardo Moschetta e Eduardo Basso; sou grato pela amizade e parceiria à tantos anos;
- Aos professores do Instituto de Informática da Ufrgs, pelos ensinamentos adquiridos;
- À Deus, pela fantástica criação.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

FA	Função de Ativação
IA	Inteligência Artificial
DoS	Denial Of Service
FPS	First Person Shooter
MLP	Multilayer Perceptron
MSE	Mean Squared Error
P2P	Peer-to-Peer
RNA	Redes Neurais Artificiais
RTS	Real-Time Strategy
MMOG	Massively Muti-player Online Games
FTLFN	Focused Time Lagged Feedforward Network

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

No presente trabalho, é testada e avaliada a aplicação de Redes Neurais Artificiais no combate de trapaças (*cheating*, em inglês) do tipo *speed cheating* em jogos *online* massivos de múltiplos jogadores, também conhecidos como MMOG (*Massively Multiplayer Online Games*). Os MMOG representam um modelo de negócio onde quantias significativas de recursos financeiros estão envolvidas, e crescem a cada dia. Os modelos para o combate de trapaças, que possam afastar jogadores de jogos ou servidores, estão localizados na camada de rede, à nível de protocolo. Analisando o estado-da-arte, constatou-se que não existem trabalhos explorando a área de Inteligência Artificial para este fim, tornando-se assim relevante o estudo de sua aplicabilidade. As Redes Neurais Artificiais foram escolhidas por terem grande poder de abstração, generalização e plasticidade. Através dos resultados obtidos comparando-se duas abordagens de arquiteturas, as redes *Perceptron* de múltiplas camadas (MLP) e as redes com atraso no tempo focadas (FTLFN), é possível constatar que é viável a utilização das mesmas para este fim, tendo-se alcançado resultados positivos no combate de *speed cheating* em MMOGs.

Palavras-chave: Inteligência artificial, redes neurais artificiais, jogos de computador, trapaças.

Neural Networks Applied to Speed Cheating Recognition in Online Computer Games

ABSTRACT

In the present work, Artificial Neural Networks are tested and evaluated in order to avoid a specific type of cheating, called Speed Cheating, in massively multi-player online games (MMOG). The MMOG represent a business model where meaningful financial resources amounts are involved, and increase each day. The models to avoid cheating, that could keep off players from games and servers, are localized in the network layer, at the protocol level. Examining the state-of-art, it was observed that research exploring the Artificial Intelligence application to this goal becomes relevant. The Artificial Neural Networks were chosen by their significant abstraction, generalization and plasticity characteristics. Through the results's comparison from two different architectures approaches, the multi layer Perceptron network (MLP) and the focused time lagged network (FTLFN), it was possible to conclude that their utilization avoiding speed cheating in MMOG is possible, once good results were found in this work.

Keywords: Artificial intelligence, artificial neural networks, computer games, cheating.

1 INTRODUÇÃO

1.1 Motivação

Jogos de computador inicialmente apareceram como sendo meramente fonte de entretenimento para as novas gerações da sociedade (YAN; RANDELL, 2005a). Entretanto, mais do que entretenimento, eles têm sido por décadas uma das maiores aplicações computacionais, gerando diversas motivações para o avanço de pesquisa em diferentes áreas. Atualmente, os jogos lideram o desenvolvimento de hardware gráfico e de software, e a indústria de jogos de computador ocupa um mercado multi-bilionário (KABUS et al., 2005).

Com o surgimento de novas tecnologias de transmissão de dados, permitindo que o acesso à Internet fosse rapidamente aumentado, jogos *online* tornaram-se uma das aplicações mais populares na rede mundial de computadores(YAN; RANDELL, 2005b), mudando a maneira que os jogos de computador são jogados. Tradicionalmente, a maior parte dos jogos de computador eram caracterizados por permitirem que as partidas fossem jogadas por usuários fisicamente próximos, ou em muitos casos, o oponente era emulado pelo próprio jogo, em que eram utilizadas técnicas de inteligência artificial para simular um adversário que parecesse humano.

Paralelamente à mudança significativa ocorrida nos jogos, a questão de segurança igualmente mudou. Para os primeiros jogos de computador, a principal questão de segurança estava relacionada à proteção de cópias, ou seja, dificultar que um jogo pudesse ser pirateado.

No caso dos jogos *online* foi criado um novo estilo de negócio, onde muitas vezes simplesmente é ignorado o fato da cópia ser pirata ou não, sendo assim, é cobrado do usuário apenas quando este se conecta em um servidor para jogar. A questão de segurança, neste novo modelo de jogo, passou a estar relacionad com possíveis comportamentos fraudulentos por parte de jogadores, através de trapaças.

Cheating, ou trapaça em português, existe desde o momento da criação dos jogos. Segundo (YAN; RANDELL, 2005c), trapaça pode ser definida como "a execução de algum ato que esteja fora dos métodos normais do jogo ou competição com uma expectativa ou esperança que isto irá trazer uma vantagem fora do normal para com o jogo."A definição da clara separação entre o que pode ser definido como sendo trapaça ou não, é uma tarefa não muito simples, pois muitos jogos oferecem uma vasta gama de personalizações que podem ser realizadas. Por exemplo, em alguns jogos, o usuário pode modificar a vestimenta de seu personagem. Esta alteração, pode dificultar que outro jogador perceba a sua localização, porém, todos os jogadores do jogo possuem esta opção de personalização, em inglês *customization*. Segundo a definição citada anteriormente, este exemplo seria considerado trapaça caso esta opção não fosse habilitada pelo jogo.

Os jogos com suporte a um número massivo de jogadores, também conhecidos como *massively muti-player online games* ou MMOG, como o jogo *World of Warcraft*, que é um dos mais populares MMOG de todo o mundo, contando com mais de seis milhões de jogadores, constitui-se um fenômeno ao contexto dos jogos eletrônicos. Este tipo de jogo multi-jogador possui duas características adicionais: envolvem um quantidade relativamente grande de jogadores simultâneos interagindo em tempo real e suportam uma simulação de estado persistente. A quantidade de jogadores pode variar de algumas centenas até centenas de milhares ou mesmo milhões de jogadores, como visto em (CHEN; MAHESWARAN, 2004).
Neste tipo de jogo, o problema de trapaça é crítico, tanto em arquiteturas centralizadas como em descentralizadas (JR. et al., 2004). Enquanto a trapaça em jogos do tipo *single player* pode criar apenas um conflito para apenas um jogador; neste modelo de jogo, como foi citado anteriormente a quantidade de jogadores que podem estar atuando, o estrago potencial causado pela trapaça de um usuário é enorme (SMED; KAUKORANTA; HAKONEN, 2001), pois além de afastar os jogadores de um determinado servidor, também afasta-os do jogo em questão.

Atualmente as técnicas e ferramentas utilizadas para o combate de trapaças em jogos *online* de computador estão concentradas na camada de rede, mais especificamente na questão de envio de mensagens, como será apresentado no capítulo 2. Porém, as técnicas utilizadas para esse fim não abrangem todos os tipos de trapaças, e nem algumas que são corriqueiramente utlizadas no dia-a-dia de jogos *online*. Dessa forma, torna-se interessante explorar outras áreas para o combate de trapaças, sendo uma delas a área de inteligência artificial. Nesta área, uma das correntes que têm sido utilizada com sucesso para diversos fins de reconhecimento de padrões, são as Redes Neurais Artificiais. O capítulo 3 irá apresentar o tema em detalhes.

1.2 Objetivos

Este trabalho apresenta como principais objetivos:

- Levantar as técnicas de trapaças mais usuais e críticas em jogos *online* de computadores;
- Estudar as técnicas e ferramentas para o combate das trapaças levantadas;
- Pesquisar as técnicas de Redes Neurais Artificiais que se adaptariam para a resolução da trapaça levantada, levando em conta as limitações do jogo em questão;
- Implementação e testes para validação dos modelos de Redes Neurais Artificiais, e;
- Avaliação dos resultados obtidos.

1.3 Organização do trabalho

O restante deste trabalho está organizado da seguinte maneira:

- *CAPÍTULO 2 - JOGOS DE COMPUTADOR:* Este capítulo apresenta detalhes sobre jogos de computador e trapaças. São apresentados tipos de jogos, a infraestrutura de desenvolvimento e as arquiteturas nas quais os jogos são disponibilizados. Em relação às trapaças, serão apresentados os tipos mais comuns, assim como as técnicas e ferramentas utilizadas atualmente para a detecção e prevenção.

- *CAPÍTULO 3 - REDES NEURAIS ARTIFICIAIS:* Este capítulo introduz a técnica de Redes Neurais Artificiais e discute a sua utilização em sistemas de reconhecimento de padrões. Será apresentado um breve histórico, as arquiteturas mais utilizadas para a detecção de padrões, bem como os algoritmos de treinamento.
- *CAPÍTULO 4 - DESENVOLVIMENTO DO SISTEMA DE DETECÇÃO:* Descreve os detalhes do projeto e do jogo no qual o sistema de detecção de trapaças será desenvolvido. Serão apresentados também aspectos de implementação do protótipo desenvolvido.
- *CAPÍTULO 5 - EXPERIMENTOS E RESULTADOS:* Neste capítulo é apresentado o conjunto de experimentos que foi realizado no âmbito de detecção de trapaças.
- *CAPÍTULO 6 - CONCLUSÕES:* O último capítulo discursa sobre a relevância dos resultados obtidos, e aponta as melhorias a serem aplicadas futuramente, em possível continuidade ao trabalho.

2 JOGOS DE COMPUTADOR

2.1 Introdução

Este capítulo apresenta uma revisão bibliográfica sobre o assunto de jogos *online* de computador. O objetivo é fazer uma breve introdução ao modelo dos jogos, e levantar as principais ameaças em relação às técnicas e modelos de trapaças e ferramentas disponíveis atualmente para o seu combate. A partir dos dados levantados neste capítulo, é possível adquirir uma noção da criticidade das trapaças nos jogos hoje em dia, e o que existe disponível em termos de pesquisa para o seu combate.

O texto deste capítulo está organizado em seis seções. Nas três primeiras seções, são introduzidos o tema de jogos de computadores, onde são apresentados os detalhes como gênero dos jogos, formas de modelo para os jogos e possíveis arquiteturas. Nas seções seguintes, são apresentados o tema de trapaças, onde são levantadas as técnicas de trapaças e as ferramentas e técnicas utilizadas hoje em dia para o combate das mesmas.

2.2 Gênero dos jogos

Atualmente, existem diversos títulos de jogos de múltiplos jogadores disponíveis, porém, eles podem ser divididos nos gêneros apresentados à seguir.

2.2.1 Jogos de tiro de primeira pessoa

Também conhecido por *First Person Shooter* (FPS) em inglês, é um gênero caracterizado pelo fato de o ambiente tri-dimensional é percebido pelo jogador através dos olhos do seu avatar no mundo virtual simulado, ou seja, o jogador é imerso no jogo. O objetivo do jogo é geralmente colecionar armas e munição em ordem de lutar contra outros jogadores. Este gênero foi criado antes dos jogos em redes, como por exemplo Wolfenstein 3D, mas com a possibilidade de se jogar contra outros jogadores *online* mudou-se rapidamente da forma tradicional de combate, para jogos orientados a missões onde o desafio seria competir diretamente contra outros jogadores humanos.

Jogos FPS de multiplos jogadores *online* são exclusivamente implementados usando-se uma arquitetura cliente/servidor com um possível servidor dedicado rodando a simulação do mundo virtual e jogadores enviando comandos através dos clientes locais (WITTENBURG, 2004). Dada a natureza dos tipo de jogo, jogos FPS tendem a ser muito sensíveis em relação ao tempo de comunicação entre todos os hosts conectados no jogo. Pesquisas recentes sugerem que jogos FPS tornam-se não-jogáveis se o atraso entre a ação e a manifestação da ação no mundo simulado for maior do que 100ms (WITTENBURG, 2004).

2.2.2 Jogos de estratégia em tempo-real

Também conhecido por *Real-Time Strategy Games* (RTS) em inglês, são jogos onde o jogador não se identifica diretamente com um simples avatar no jogo, ao invés disto, ele comanda um grupo de unidades as quais são observadas em outra perspectiva (WITTENBURG, 2004). Jogos RTS são geralmente construídos com o objetivo de colecionar recursos, construir uma infra-estrutura, e usando isto para atacar e derrotar jogadores oponentes. Portanto, enfatizam conceitos de plano estratégico e micro-gerenciamento (GARFINKEL et al., 2003).

Jogos RTS não são sensíveis ao atraso como os jogos FPS, porque a granularidade do controle requerido sobre uma unidade em particular é significativamente pequena (CHEN et al., 2004). Modernos jogos enfatizam níveis de Inteligência Artificial que permitem unidades realizarem simples tarefas sem a intervenção do jogador. A arquitetura predominante dos jogos RTS é simular o jogo em cada um dos computadores conectados e somente realizam comunicações por troca de comandos realizados pelo respectivo jogador local. Isto duplica a computação requerida, mas reduz significantemente a sobrecarga de comunicação (WITTENBURG, 2004).

2.2.3 Jogos *online* massivo de múltiplos jogadores

Também conhecido por *Massively Muti-player Online Games* (MMOG em inglês), é um gênero relativamente novo, enquanto que os jogos RPG já existiam há vários anos. O conceito de MMORG é diferente no modo em que é dado ênfase à intereção com outros jogadores reais ao invés de resolver uma busca potencialmente muito complicada (WITTENBURG, 2004). Os jogadores também são imersos no jogo e enxergam o mundo virtual através dos olhos do seu avatar, ou tem uma perspectiva geral pertencente ao seu avatar.

Uma característica deste gênero de jogo é o fato que a simulação do mundo virtual é gravada no servidor central mesmo se o jogadores não estão conectados ao jogo, o que permite ao jogador executar objetivos de longo prazo (WITTENBURG, 2004). As interações entre jogadores possuem uma severidade menor quando comparado ao gênero de FPS.

2.3 Modelando jogos

Os jogos de computador atualmente, podem ser modelados consistindo basicamente em três camadas, como mostrado no seção 2.3.1. Já os jogos *online*, possuem características próprias, e será mostrado um modelo básico na seção 2.3.2.

2.3.1 Um modelo geral para jogos

Segundo (WITTENBURG, 2004), qualquer jogo pode ser modelado como consistindo de três camadas lógicas, como mostrado em 2.1.

Gameplay : As interações entre os jogadores humanos, incluindo as regras, que não forçam pela implementação do jogo. Este é o conceito de "jogar pôquer"para um jogo de carta, ou um contexto muito específico do jogo para um jogo *online* de múltiplos jogadores.

Implementação : Os recursos e dispositivos físicos pelas quais o jogo é jogado. Isto seria as cartas físicas para um jogo de carta, ou o *software* cliente/servidor para um

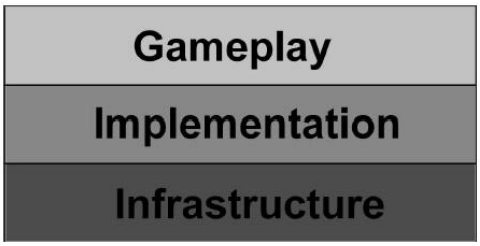


Figura 2.1: Três camadas lógicas de um jogo.

jogo *online*.

Infraestrutura : A infraestrutura necessária para jogar o jogo em uma primeira instância, isto é mesa e cadeiras para um jogo de pôquer real, ou um computador e uma rede de dados para um jogo *online*.

Faz-se necessário notar que jogos *online* de computador contam pesadamente com a camada de implementação para estabelecer características e regras do jogo, enquanto que jogos de múltiplos jogadores sem rede podem fazer isto na camada de *gameplay*, pelo motivo de todos os jogadores estarem fisicamente presentes na maior parte do tempo.

2.3.2 Um modelo para jogos *online*

Atualmente, existem duas arquiteturas predominantes para a camada de implementação no jogos *online* de múltiplos jogadores. De um lado têm-se o modelo cliente/servidor, como implementado por jogos FPS. Nesta arquitetura o servidor possui a completa representação do mundo simulado, o "estado do jogo", enquanto que os clientes interagem com este estado enviando comandos para o servidor e somente recebendo informação que é disponível apenas para eles baseada em suas limitadas visões deste mundo simulado (WITTENBURG, 2004).

Como uma alternativa, têm-se a arquitetura ponto a ponto, do inglês *peer-to-peer* (P2P), que distribui o estado do jogo entre todos os jogadores, os quais executam uma parcial, ou completa simulação do mundo do jogo (WITTENBURG, 2004). Esta simulação mantém um consistente estado global pela troca de comandos que cada jogador envia para eles.

Independentemente da arquitetura escolhida, tema que será abordado em 2.4, vários conceitos são compartilhados entre todos os jogadores *online*:

1. Há um estado de jogo global. Isto define o estado atual do mundo virtual simulado.

2. Cada jogador tem acesso à um subconjunto do estado do jogo. Isto é a visão que o jogador possui do jogo.

3. Jogadores manipulam o mundo virtual enviando comandos para a simulação. Quais comandos são considerados legais depende da situação corrente.

4. Não exitem outros meios, senão os listados anteriormente, para interagir com o mundo simulado.

Embora todos os tipos de jogos possuam maior ou menor requerimentos de tempo-real e tentem apresentar ao jogador como uma simulação contínua, atrasos nos efeitos dos

jogos são geralmente implementados por unidades de tempo discretas (WITTENBURG, 2004). Propriedades de tempo-real são aproximadas fazendo estes períodos de tempo relativamente curtos. Por exemplo, para jogos FPS a duração de um ciclo de jogo será na mesma ordem de magnitude da taxa de atualização de vídeo, aproximadamente 30 ciclos por segundo.

2.4 Arquitetura para jogos massivos de múltiplos jogadores

Quando se projeta um jogo do tipo MMOG, existe um número de possíveis arquiteturas à serem escolhidas, cada uma com seus pontos positivos e negativos. Ao se escolher uma arquitetura, deve ser levado em consideração quatro atributos, que são: **escalabilidade** (quão fácil a arquitetura pode suportar clientes adicionais respeitando a quantia de recursos para acomodá-los), **confiabilidade** (quão bem o mundo virtual do jogo pode reestabelecer-se caso um servidor da rede caia), **segurança** (quão bem a arquitetura pode detectar e lidar com trapaças), e **persistência** (quão bem a arquitetura preserva o estado do mundo virtual do jogo e/ou os dados do cliente quando este decide sair ou retornar ao jogo).

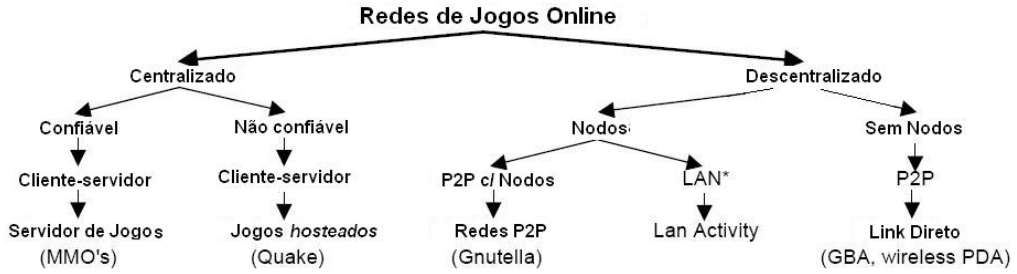


Figura 2.2: Taxonomia de arquitetura de jogos por categoria.

Na figura 2.2, onde é apresentada uma taxonomia de arquiteturas, temos o modelo P2P. Aqui, todos os computadores atuam como cliente e servidor ao mesmo tempo. O quesito escalabilidade é o melhor aqui, pois não há um simples ponto para o qual todos os dados são direcionados, o que poderia criar um gargalo na rede quando muitos jogadores estiverem envolvidos (BROMMELHOFF; KING; CLAYPOOL, 2005). Igualmente, a confiabilidade é excelente, porque se um jogador abandonar o jogo, não irá causar o término do jogo para os outros jogadores, empregando-se estruturas centralizadas. A segurança é puramente baseada num sistema de honra; qualquer jogador à qualquer momento pode dar à sí próprio uma vantagem injusta sobre os outros jogadores desde que não exista um mediador para averiguá-los, isto torna-se um problema significativo em algumas situações (BROMMELHOFF; KING; CLAYPOOL, 2005). Finalmente, não há persistência com um projeto P2P básico.

Outra arquitetura que existe para a contrução de jogos é a cliente/servidor. Neste caso exite um servidor no qual todos os clientes estão conectados. Este servidor atua como sendo uma entitade "poderosa e inteligente"através da qual todos os dados sobre o jogo são enviados. Assim, têm-se como resultado um aumento drástico em segurança, além de oferecer a possibilidade de persistência dos dados. De modo oposto, entretanto, esta arquitetura sozinha é fracamente escalável, já que o servidor pode tornar-se um foco de gargalo e um servidor único não é confiável (BROMMELHOFF; KING; CLAYPOOL, 2005).

Um passo à frente da arquitetura básica P2P, é a adição de super-nós (*super-nodes*). Cada computador ainda atua como cliente, porém ao invés de todos constituírem-se servidores, somente certos computadores são servidores em um dado instante, apesar de que cada computador é um possível candidato à ser um (BROMMELHOFF; KING; CLAYPOOL, 2005). Os computadores super-nós são mudados esporadicamente. A escalabilidade continua excelente pelas mesmas razões. Esta arquitetura P2P oferece alguma persistência enquanto os super-nós estiverem ativos, entretanto, isto também introduz leves problemas com confiabilidade - se cada super-nó cair, o jogo não poderá continuar. Esta arquitetura apresenta algumas vantagens sobre a arquitetura P2P básica, primeiramente na questão de segurança. Enquanto o P2P com super-nós oferece alguma forma de detecção de trapaças, já que aqueles que são super-nós poderiam potencialmente abusar dos seus poderes ou olhar de outra maneira a questão de trapaça (BROMMELHOFF; KING; CLAYPOOL, 2005).

Uma terceira variante da P2P é a adição de nós confiáveis (*trusted nodes*). Novamente, todos os computadores são clientes, mas somentes os confiáveis podem ser servidores (BROMMELHOFF; KING; CLAYPOOL, 2005). Isto oferece vantagens em segurança e persistência, desde que os computadores confiáveis estão associados a pessoas contratadas pela companhia que está rodando o jogo. Porém, isto faz a arquitetura menos escalável do que as versões anteriores de P2P, porquê com a adição de mais clientes serão necessários mais nós confiáveis (BROMMELHOFF; KING; CLAYPOOL, 2005). Por último, a confiabilidade é um pouco menor, porque os servidores são limitados, estes podem cair acidentalmente.

Por final, têm-se a arquitetura fazenda de servidores distribuídos. Neste caso, temos múltiplos computadores, todos servidores, trabalhando juntamente como um sistema distribuído (BROMMELHOFF; KING; CLAYPOOL, 2005). Todos os outros computadores conectam-se neste sistema como clientes. Esta arquitetura tem uma grande capacidade em termos de segurança e persistência, visto que todos os dados podem ser salvos para os servidores (BROMMELHOFF; KING; CLAYPOOL, 2005). Entretanto, com uma grande quantidade de dados sendo enviados para o servidor, isto pode gerar um gargalo e a necessidade de servidores adicionais poderá ser necessária, especialmente se o MMG se tornar popular. O quesito confiança não é alto, visto que novamente, os servidores podem cair acidentalmente.

Abaixo, na tabela 2.1, é feita uma comparação entre todas as arquiteturas analisadas.

Tabela 2.1: Comparação entre arquiteturas de jogos

Nome	Escalabilidade	Confiabilidade	Segurança	Persistência
P2P	excelente	excelente	"sistema de honra"	fraca
P2P com super-nós	excelente	boa	"sistema de honra"	fraca
P2P com nós confiáveis	boa	boa	fraca	fraca
Fazenda de servidores distribuídos	fraca	fraca	excelente	excelente
Cliente-servidor	fraca	fraca	excelente	excelente

2.5 Trapaça em jogos

Nesta seção pretende-se realizar o estudo propriamente dito em relação a trapaças em jogos *online* de computador. Primeiramente, é mostrada uma visão geral sobre o ato de trapacear, vendo-se também questões legais referentes ao tema. Por fim, é feito um estudo

de natureza técnica, analisando os tipos mais comuns de trapaça e possíveis classificações atribuídas às diversas técnicas de trapaça.

2.5.1 O que é trapaça

Definir os limites entre o que é considerado trapaça e o que não é, torna-se uma árdua tarefa. Muitos jogos oferecem uma extensa variedade de opções configuráveis e personalizações (JR. et al., 2004). Assim, algumas destas configurações poderiam ser definidas como sendo trapaças. Por exemplo, em alguns jogos, um jogador pode mudar a cor da vestimenta do seu personagem com o objetivo de se camuflar em relação ao ambiente. Esta alteração dá ao jogador a vantagem de dificultar sua detecção; porém todos os jogadores possuem a opção de usar esta configuração (WEBB; LI, 2004).

Usando o exemplo citado anteriormente, trapaça pode ser definido como sendo qualquer ação tomada por um jogador para obter uma injusta vantagem sobre outros jogadores (WEBB; LI, 2004). Seguindo esta definição, o exemplo anterior não seria caracterizado como trapaça porquê a opção de cor de vestimenta é parte do jogo originalmente.

2.5.2 Visão legal sobre trapaça

Trapaças e ações relacionadas podem, sob certas circunstâncias, ser considerados como crime punidos pela lei (ZETTERSTRöM, 2005). Nestas instâncias, ou seja, violação de propriedade intelectual e crime cibernético na forma de acesso não autorizado em servidores e computadores, o melhor que pode ser feito é reportá-los para as autoridades. Naturalmente, quanto mais evidências forem entregues para o executor e o policiamento será melhor.

2.5.3 Questões éticas sobre trapaça

Estudos foram realizados sobre a questão moral da trapaça. Segundo descrito em (KIMPA; BISSET, 2005), uma teoria ética fundamental diz que: *a visão liberal de tentar obter a própria felicidade ao passo de não diretamente atrapalhar a possibilidade de outras pessoas buscarem a sua própria.* A partir disto, torna-se evidente que um caso de trapaça em jogo vai de encontro à esta teoria. Um problema imposto pelos jogos *online*, é o fato da enorme distância que separa os jogadores. Dessa forma, quando um participante comete trapaças, ele não enxerga o oponente como uma pessoa física da mesma forma que ele se vê, mas sim como um personagem fictício.

Questões dessa natureza, abordadas em (KIMPA; BISSET, 2005), mostram que a trapaça é como um ciclo vicioso, onde após ter sido realizada a primeira vez contra um ou grupo de jogadores, esta prática tende a ser repetida por estes mesmo participantes que foram lesados numa primeira vez.

2.6 Técnicas de trapaça

As técnicas de trapaça são tipicamente implementadas de três maneiras: modificação de clientes, modificadores de OpenGL, e arquivos codificados. Modificação de clientes se refere ao fato de inserir código diretamente no jogo enquanto está carregado na memória do computador (WEBB; LI, 2004). Modificadores de OpenGL modificam a funcionalidade dos *drivers* OpenGL, assim, os gráficos são desenhados de uma forma diferente (WEBB; LI, 2004). Os arquivos codificados são tipicamene bibliotecas de lincagem dinâmicas (DLL) ou arquivos de configuração que são modificados para alterar o

comportamento dos jogos (WEBB; LI, 2004). A seguir, serão discutidas categorias que podem ser usadas para descrever as várias formas de trapaça em jogos *online*.

2.6.1 Exploração de falhas

Os jogos, assim como qualquer *software*, não estão imunes às falhas (*bugs*) de programação. Infelizmente, em alguns casos, estas falhas podem ser utilizadas por trapaceiros a fim de obter vantagens sobre os outros jogadores.

2.6.1.1 Má randomicidade

Muitos jogos se apoiam em eventos gerados randomicamente para garantirem lealdade (WEBB; LI, 2004). Por exemplo, jogos de carta tais como bridge e pôquer requerem que o montante de cartas seja embaralhado perfeitamente antes que cada uma seja entregue, para que nenhum jogador tenha vantagem sobre outro. Tradicionalmente, jogos de computador efetuam este embaralhamento usando um gerador de números pseudorandômico (WEBB; LI, 2004). Caso este gerador de números seja defeituoso, ou a implementação do algoritmo seja incorreta, as cartas não serão distribuídas de uma maneira randômica, assim, a justiça no jogo não será preservada.

2.6.1.2 Escapada

A maior parte dos jogos *online* associam uma classificação à cada jogador participante, o qual serve como um indicador de nível de habilidade do jogador (WEBB; LI, 2004). Esta classificação é tipicamente calculada baseando-se no registro de vitórias, derrotas e jogos inacabados. Entretanto, alguns sistemas não levam em consideração jogos inacabados ao calcularem esta classificação. Como resultado, trapaceiros podem se desconectar do jogo no qual eles estão perdendo sem afetar negativamente suas classificações (WEBB; LI, 2004).

2.6.1.3 Colisão de comando

Em jogos *online*, se a rede de comunicação se encontra em um significativo atraso, o jogo deve postergar a próxima rodada até que a comunicação possa continuar normalmente (WEBB; LI, 2004). Entretanto, enquanto o jogo é suspenso, os jogadores ainda estão hábeis para interagirem com a interface do jogo. Conseqüentemente, um jogador pode executar vários comandos enquanto a próxima rodada está sendo postergada. Quando o jogo retornar, ele irá processar todos os comandos enviados pelo jogador (mesmo duplicados) como se eles fossem executados numa única vez.

2.6.2 Aumento da habilidade

Vários jogos requerem reações rápidas e precisas para o seu sucesso. Por exemplo, jogos FPS tais como *Counter-Strike* e *Quake III Arena* são dominados por jogadores que possuem rápidos reflexos e grande coordenação olho-mão. Tipicamente, estes talentosos jogadores passam inúmeras horas aperfeiçoando suas habilidades, porém trapaceiros querem resultados com o menor esforço possível. Assim, trapaceiros freqüentemente fazem uso de técnicas para aumentar suas habilidades em jogos a fim de competirem no mesmo nível de legítimos bons jogadores. Os seguintes métodos exemplificam tais técnicas.

2.6.2.1 Trapaça de pontaria

Em um jogo FPS, o objetivo é mirar e atirar em outros jogadores. O aspecto mais importante deste processo é o ato de mirar, pois o tiro na cabeça *head-shoot*) em outro jogador é muito mais letal do que o tiro no braço ou na perna, por exemplo. Assim, jogadores com a melhor mira são consistentemente os mais bem sucedidos nestes tipos de jogos. Trapaceiros desejam esta habilidade de mira, mas eles não desejam perder tempo praticando (YAN; RANDELL, 2005c). Como resultado, eles usam uma trapaça de pontaria (*aim hack*) para obter habilidades de mirar. Dois tipos de *hack* de mira existem atualmente, mas ambos provêem para o trapaceiro uma mira perfeita. A primeira trapaça de pontaria atua como um mandatário entre o jogo do trapaceiro e o servidor de jogo (WEBB; LI, 2004). Quando o trapaceiro tenta atirar em outro jogador, este mandatário insere comandos adicionais ao jogo para garantir que o trapaceiro está mirando diretamente ao jogador oponente mais próximo (YAN; RANDELL, 2005a). A segunda trapaça de pontaria é de fato adicionado ao jogo, mas provê o mesmo tipo de funcionalidade que o mandatário oferece. Entretanto, a segunda trapaça de mira pode ser configurado para disparar automaticamente em outros oponentes.

2.6.2.2 Trapaça de velocidade

A mira é muito importante em jogos FPS, mas esquivar-se da mira de outros é igualmente importante. Uma mira de um jogador é irrelevante se o jogador é morto antes de atirar. Assim, para evitar de ser alvo, bons jogadores pulam ao redor do ambiente, esquivando-se de balas e frustrando outros jogadores. No entanto, estas manobras requerem grande habilidade e prática - duas coisas que trapaceiros querem obter rapidamente (WEBB; LI, 2004). Em ordem de evitar com sucesso ser alvo nestes tipos de jogos, trapaceiros freqüentemente empregam uma trapaça de velocidade (*speed cheating*) o qual aumenta a taxa que eles podem se mover no jogo (YAN; RANDELL, 2005c). A lógica por trás deste tipo de trapaça é simples: quanto mais rápido o alvo se mover, mais difícil será para atirar.

2.6.2.3 Trapaça anti-granada

Muitos jogos contém itens que podem ser usados temporariamente para prejudicar as habilidades de outros jogadores. Por exemplo, no jogo *Counter-Strike*, os jogadores tem a opção de usar granadas de luz e de fumaça para, momentaneamente, nublar a visão do oponente. Para combater esta situação, trapaceiros utilizam uma trapaça anti-granadas (*anti-grenade hack*) para remover os obstáculos de visão da tela (WEBB; LI, 2004). Conseqüentemente, esta trapaça torna o trapaceiro imune a granadas de luz, de fumaça, e outros tipos de ítens que servem para prejudicar as habilidades de legítimos jogadores.

2.6.3 Negação de serviço

A maior parte de trapaças são usadas para dar habilidades injustas durantes o decorrer do jogo. No entanto, a proposta de trapaças de negação de serviço, do inglês *denial of service* (DoS), é negar acesso ao jogo para jogadores legítimos. Trapaceiros usam estas técnicas por vários motivos. Primeiro, se um trapaceiro implementar um DoS contra um oponente enquanto o jogo está em andamento, o trapaceiro pode forçar o oponente a se desconectar, obtendo uma fácil vitória neste processo. Segundo, uma trapaça de DoS pode ser usada para prevenir que um inimigo ou jogador superior de acessar o jogo, permitindo que o trapaceiro seja impedido de jogar contra jogadores específicos. São exibidos em

seguida vários tipos de DoS.

2.6.3.1 Trapaça de atraso

Em jogos *online*, a latência da rede é fundamentalmente importante porque determina quão rápido os comandos de um jogador serão recebidos pelo servidor de jogo (WEBB; LI, 2004). Assim, quanto maior a latência da rede presente no jogo *online*, menor será o tempo de resposta que o jogo terá para este jogador. Como resultado, os desenvolvedores de jogo atentam para minimizar a quantidade de informação trocada entre os jogadores e o servidor de jogo para evitar problemas de latência. Entretanto, trapaceiros podem introduzir artificialmente latência na rede inundando a conexão de outro jogador à rede com tráfego sem sentido (WEBB; LI, 2004). Este tráfego adicional serve com dois propósitos ao trapaceiro. Primeiro, isto diminui a percepção de resposta do jogo para jogadores legítimos. Conseqüentemente, jogadores não-trapaceiros podem sentir a necessidade de abandonar o jogo devido ao excesso de latência e tempo de resposta muito lento. Segundo, o tráfego adicional poderia potencialmente diminuir a conexão do jogador não-trapaceiro à ponto do jogo parecer sem resposta. Isto resultaria no jogador sendo desconectado do jogo (WEBB; LI, 2004). Ambos cenários resultam em uma fácil e injusta vitória do trapaceiro.

2.6.3.2 Ataque de login inválido

Para acessar a maior parte do jogos *online*, os jogadores precisam logar no servidor de jogo com um único nome de usuário e senha. Este processo de *login* autentica o jogador para o servidor, prevenindo trapaceiros de acessar contas que não pertencem à estes (WEBB; LI, 2004). Como precaução adicional de segurança, alguns jogos *online* somente permitem um certo número de tentativas de *login* antes da conta ser temporariamente bloqueada. Esta precaução é tipicamente implementada para ajudar a prevenir ataques de quebra *online* de senha, tais como ataque de dicionário (*dictionary attack*), **o qual blablabla**. No entanto, esta função de segurança permite a um trapaceiro bloquear jogadores não-trapaceiros fora de suas contas (WEBB; LI, 2004). Propositadamente, quando um trapaceiro tenta realizar a autenticação em uma conta com uma senha inválida, após um determinado número de tentativas, ele estará hábil a bloquear aquela conta. Assim, o trapaceiro pode facilmente negar não-trapaceiros a acessarem suas contas.

2.6.4 Conhecimento de informação secreta

Em muitos jogos *online*, o computador do jogador possui conhecimento de tudo o que está acontecendo no jogo corrente. Por exemplo, em jogos RTS como *StarCraft* e *WarCraft*, o computador do jogador sabe informações sobre outros jogadores no jogo mesmo se eles não podem serem vistos pelo jogador (WEBB; LI, 2004). O jogador é suposto ser inconsciente desta informação até que o jogador encontre os outros jogadores. O computador do jogador possui esta informação porque permite ténicas como contagem de mortes (*dead reckogning*), e isto é mais eficiente do que ter o servidor constantemente atualizado com informação conhecida pelo computador do jogador (YAN; RANDELL, 2005c). No entanto, maliciosamente à respeito da natureza destas informações, os trapaceiros são habeis para utilizá-las a fim de obter vantagem.

2.6.4.1 Trapaça de mapa

Em jogos RTS, um jogador controla alguma quantia de personagens os quais são usados para completar vários objetivos - tipicamente a destruição de outros personagens. Estes jogos oferecem aos jogadores um mundo no qual seus personagens interagem, mas usualmente, cada visão de mundo do jogador é restrita às áreas nas quais o jogador explorou (WEBB; LI, 2004). Se uma sessão do mundo não foi explorada pelo jogador, é chamada de "neblina de guerra"(*fog of war*). Nesta situação, a neblina serve para ocultar a informação secreta - as áreas do mapa que não foram previamente exploradas pelo jogador. Para explorar este cenário, trapaceiros empregam uma trapaça de mapa (*map hack*), o qual remove a "neblina de guerra"independentemente se o trapaceiro explorou o mapa inteiro ou não (YAN; RANDELL, 2005b). Fazendo isto, trapaceiros estão hábeis para obter vantagem injusta porque eles terão conhecimento de informações secretas.

2.6.4.2 Trapaça de parede

Em jogos FPS, os jogadores estão cercados por vários obstáculos (paredes, caixas, etc), e eles devem manobrar taticamente ao redor do mapa para encontrar e atirar em outros jogadores. Num jogo normal, os jogadores são hábeis somente para verem outros jogadores que estão em seu campo de visão. Isto significa que se os jogadores estão escondido atrás de obstáculos, eles não devem ser visíveis para outros jogadores. Jogos mostram um campo de visão desenhando cada cena de trás para frente (WEBB; LI, 2004). No entanto, se outros jogadores estão escondidos atrás dos obstáculos, estes são desenhados no topo deles, parecendo como se eles não estivessem ali. Um trapaceiro é hábil para explorar esta situação usando uma trapaça de parede (*wall hack*) o qual desenha os obstáculos transparentemente (WEBB; LI, 2004). Assim, os jogadores que estão escondidos atrás dos obstáculos não estarão mais escondidos porque apareceram transparentes.

2.7 Ferramentas anti-trapaça

Nesta seção serão abordadas algumas técnicas e ferramentas utilizadas para o combate de trapaças em jogos *online* de computador. A maior parte das técnicas são realizadas em nível de rede, enquanto outras usam métodos intrusivos de vasculhamento de memória do computador cliente.

2.7.1 Protocolo NEO

O projeto do protocolo *New-Event Ordering* (NEO) proposto por (GAUTHIERDICKEY et al., 2004) tem como um de seus alvos aeliminação de alguns tipos de trapaças. No NEO, o tempo é dividido em intervalos iguais, denominados rodadas (*rounds*), no qual o jogador envia uma atualização para todos os outros jogadores. Cada atualização é encriptada, e no seguinte *round*, cada jogador envia a chave para a próxima atualização para todos os outros jogadores (GAUTHIERDICKEY et al., 2004).

O protocolo usa estas rodadas com o intuito de obrigar o máximo atraso para que qualquer jogador possa enviar sua atualização. Atualizações tardias são consideradas inválidas, a menos que a maioria dos outros jogadores tenham recebido as atualizações. Cada mensagem contém um rótulo de tempo (*time-stamp*), uma assinatura, a informação de atualização encriptada, a chave da rodada anterior, e um vetor de bits das mensagens recebidas das rodadas anteriores. A consistência é alcançada através de um mecanismo distribuído de votação. Um jogador vota positivo para outro jogador se a atualização dos

outros jogadores foi recebida no tempo, caso contrário, é votado negativo (GAUTHIER-DICKEY et al., 2004). Uma atualização somente é considerada válida se a maioria dos jogadores enviou voto positivo. Em cada rodada os jogadores contam os votos que eles tem e decidem quais atualizações serão consideradas válidas.

2.7.1.1 Trapaças evitadas

As trapaças evitadas pelo NEO são relacionadas à seguir:

Trapaça de atraso : NEO combate esta trapaça através do uso de rodadas de tamanho limitados. Assim, atualizações tardias são simplesmente ignoradas;

Trapaça de *timestamp* : NEO previne esta trapaça novamente através das rodadas de tamanho limitado. Uma vez que a rodada passa, um jogador não pode mais submeter um movimento para a rodada, então é impossível receber uma atualização antes de submeter o movimento da rodada anterior;

Trapaça de inconsistência : NEO evita esta trapaça através do uso de assinaturas digitais e comparação de estados. Jogadores periodicamente auditam o estado do jogo realizando uma comparação de estados. Quando dois jogadores descobrem diferentes estados, os passos dos pacotes que eles receberam podem ser usados como evidências contra um trapaceiro;

Trapaça de conluio : NEO combate o conluio em nível de protocolo. Primeiro, NEO pode ajustar o valor da maioria suficientemente alto para prevenir conluio. Segundo, cada jogo pode ser distribuído com uma chave individual que é registrada para jogar. O processo de registro pode identificar jogadores e assinaturas digitais das chaves do jogo para prevenir que um jogador crie múltiplas identidades artificialmente a fim de controlar injustamente a quantidade de votos enviados.

2.7.2 Disseminação do estado

Um dos tipos comuns de trapaça é o acesso a estados do jogo, os quais não são permitidos de serem expostos aos jogadores. Isto inclui acessar informações estáticas (mapas) de uma simulação de ambiente ou estados dinâmicos (posição de outros jogadores). O conhecimento destas informações constitui-se em grande vantagem para o jogador.

A seguir, serão mostradas técnicas de disseminação destas informações.

2.7.2.1 Pré-carregamento

Nesta estratégia, também conhecida por *eager loading*, os estados estáticos do jogo são carregados no lado do cliente no início de uma sessão de jogo, estados dinâmicos são espalhados para todos os clientes após o servidor ter suas atualizações.

Com esta estratégia, o conjunto de conhecimento inclui todos os estados no servidor, e a taxa de visão de conhecimento ou seja, a quantidade de informações que o cliente possui do jogo como um todo, é grande no início do jogo. Com o andamento do mesmo, a taxa gradualmente reduz com mais estados no conjunto de conhecimento tornando-se membros do conjunto de visão (LI et al., 2004).

2.7.2.2 Carregamento Sob Demanda

Nesta estratégia, do inglês *on-demand loading*, o servidor apenas carrega o conjunto de visão (quantidade de informações do jogo) do cliente (LI et al., 2004). Sempre que

a visão do cliente expande, o cliente envia uma requisição para o servidor, este então responde com células do mapa e os estados dos objetos, caso haja objetos na célula em questão.

Com esta estratégia, o cliente possui praticamente nenhum conhecimento, e a taxa de conhecimento é muito baixa. Entretanto, esta técnica de disseminação de estado introduz atrasos para o cliente. A cada vez que a visão do cliente expandir, é necessário enviar uma requisição. O jogo somente pode seguir em frente quando a resposta retornar. Este atraso pode afetar a exibição, e se for muito grande, pode se tornar notável e impraticável para os jogadores.

Comparado com a técnica de pré-carregamento, o consumo do tráfego do servidor aumenta caso requisições explícitas sejam usadas (LI et al., 2004). Entretanto, a visão dos clientes expande somente quando um dos objetos do cliente se move através da sua visão.

2.7.2.3 Pré-carregamento Sob Demanda

Nesta terceira estratégia, conhecida por *on-demand preloading*, o servidor adivinha a expansão de visão de cada cliente e envia as células e os objetos associados, caso existirem, antes das requisições dos clientes (LI et al., 2004). Por pré-carregar as informações para os clientes, esta estratégia ajuda o cliente a reduzir o atraso introduzido pela técnica de pré-carregamento.

2.7.3 *PunkBuster*

Esta ferramenta foi a primeira tentativa de prevenção de trapaças pelo lado do cliente. Criada na metade do ano de 2000 e hábil na detecção de algumas trapaças existentes na época No entanto, poucos usuários aceitavam usar uma ferramenta sendo executada em *background* enquanto jogavam o jogo.

A ferramenta, a qual hoje é comercialmente produzida, é instalada no cliente e continuamente monitora a máquina do jogador com o objetivo de detectar sinais de aplicações trapaceiras (SMITH, 2004). Outra funcionalidade da ferramenta, é o envio de uma imagem (*screenshot*) da memória da máquina do cliente e enviá-la ao servidor para análise. O PunkBuster é um mecanismo que limita a autonomia dos jogadores por adicionar um custo em ordem de formar um subgrupo no qual os membros podem confiar nos demais (SMITH, 2004).

2.7.4 Uso de Juíz

Esta forma de controle de trapaças, especialmente às que se referem ao aumento de habilidade por parte de algum jogador, leva em consideração a honestidade de algum determinado grupo de jogadores. Ao detectarem que existem um jogador trapaceiro em uma rodada do jogo, é possível que mu ou mais jogadores informe ao servidor do jogo que existe alguém burlando as regras. Estes votos são acumulados no servidor central do jogo, e ao seu término, é feita a intervenção por parte de um **juíz** que irá analisar os *log´s* do possível fraudador, tomando assim alguma providência, como por exemplo, banir este jogador do servidor.

3 REDES NEURAIS ARTIFICIAIS

3.1 Introdução

Neste capítulo, é apresentado o tema das Redes Neurais Artificiais (RNA), arquiteturas compostas por unidades que imitam o funcionamento de neurônios, treinadas por algoritmos de aprendizagem supervisionada ou não-supervisionada. As RNA funcionam como aproximadores universais, freqüentemente utilizadas como classificadores.

O capítulo está organizado em cinco seções. Na primeira seção, é apresentada a inspiração biológica das redes neurais artificiais. Na seção seguinte, é apresentado o modelo elementar de neurônio artificial, o *Perceptron*. Após, um modelo de rede mais complexa é abordado. Na seção seguinte, é apresentada uma arquitetura de rede neural temporal, usada para problemas que consideram a variável *tempo* para ser analisada. Na seção final, são apresentadas aplicações em que são utilizadas as redes neurais artificiais para classificação de comportamentos, na área da segurança da informação.

3.2 Inspiração biológica

O cérebro humano constitui-se em uma das maiores façanhas da evolução. Cientistas estão apenas no início do entendimento de algumas complexas funções do cérebro. O estudo do cérebro de mamíferos e a aspiração do homem de emular a sua funcionalidade tem conduzido para um campo relativamente novo de pesquisa, conhecido como Inteligência Artificial, ou simplesmente IA.

As funções básicas de uma RNA são tentar emular a funcionalidade básica do cérebro de mamíferos para realizar funções complexas que comumente sistemas computacionais tradicionais são incapazes de fazer. Em resumo, RNAs são uma tentativa de simular o elemento de processamento de um cérebro biológico, conhecido por neurônio.

Redes neurais artificias podem realizar as seguintes tarefas, entre outras: (SHET, 2004):

- Reconhecer alguma coisa que nunca foi vistas antes;
- Classificar objetos em classes, baseados na generalização do treinamento;
- Comparar padrões, e;
- Predizer o futuro, a partir da extração de um padrão de atividades do passado.

3.2.1 Analogia com o cérebro humano

O exato funcionamento do cérebro humano continua misterioso para os membros da comunidade científica. Entretanto, existem certos aspectos que são melhor compreendidos, mais especificamente as células chamadas de **neurônios**. Os neurônios são as células que provêem a habilidade de pensar, lembrar e experimentar diversas sensações (JAIN; MAO, 1996). O número de neurônios, que acredita-se existirem no cérebro humano, é de aproximadamente 100 bilhões. Cada um desses neurônios é capaz de ter entre 1000 e 20000 conexões, porém o usual é terem de 1000 a 10000 conexões. Presume-se que o extraordinário poder do cérebro humano venha do vasto número de neurônios e suas conexões (JAIN; MAO, 1996).

Neurônios individuais são muito complexos e existem aproximadamente 100 diferentes classes de neurônios, os quais realizam diversas tarefas (SHET, 2004). Juntos, estes neurônios e suas conexões provêem para os seres humanos capacidades intelectuais.

A estrutura básica do cérebro então provê inspiração para os componentes básicos em que as redes neurais artificiais são modeladas. Existe uma forte similaridade entre o neurônio artificial e o biológico, e assim pode ser encontrada uma réplica biológica para um componente em um neurônio artificial. Pesquisa em redes neurais têm, de certa forma, muito de sua terminologia em estudos de conceitos envolvidos na neurociência. RNAs entretanto tentam emular as funcionalidades dos neurônios do cérebro, para construir máquinas que são programadas para resolver complicadas tarefas. Quando neurônios biológico e artificial são comparados entre sí, as similaridades entre os dois tornam-se evidentes. A figura 3.1 apresenta o desenho de um neurônio biológico humano.

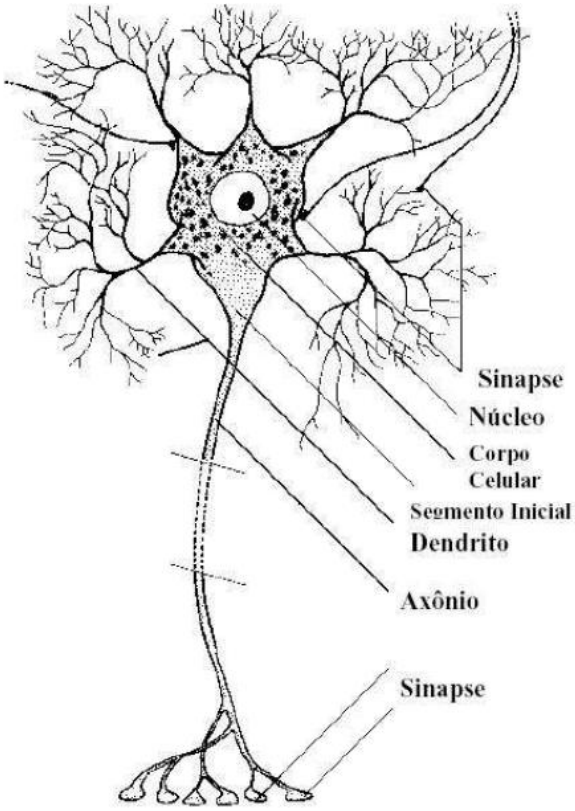


Figura 3.1: Neurônio biológico (OSORIO; BITTENCOURT, 2000).

Um neurônio biológico é formado por três partes principais, que são uma árvore dendrítica, um corpo celular e um prolongamento fino e longo, chamado de axônio. Os

dendritos são as entradas pelas quais o neurônio recebe os impulsos elétricos provenientes de outros neurônios, e o axônio é o meio por onde os sinais elétricos saem do neurônio em direção aos outros neurônios. A ligação existente entre os dois neurônios, conhecida como sinapse, é que irá influenciar sobre a transmissão do sinal recebido, podendo atuar de forma a atenuar ou amplificar os impulsos elétricos recebidos (HAYKIN, 1999).

Baseado no funcionamento do neurônio biológico, foi desenvolvido o neurônio formal, que é uma simplificação matemática que tenta reproduzir as principais características dos neurônios humanos referentes a forma em que se processa a aquisição de conhecimentos. A figura 3.2 mostra um neurônio artificial, que é utilizado nas RNAs.

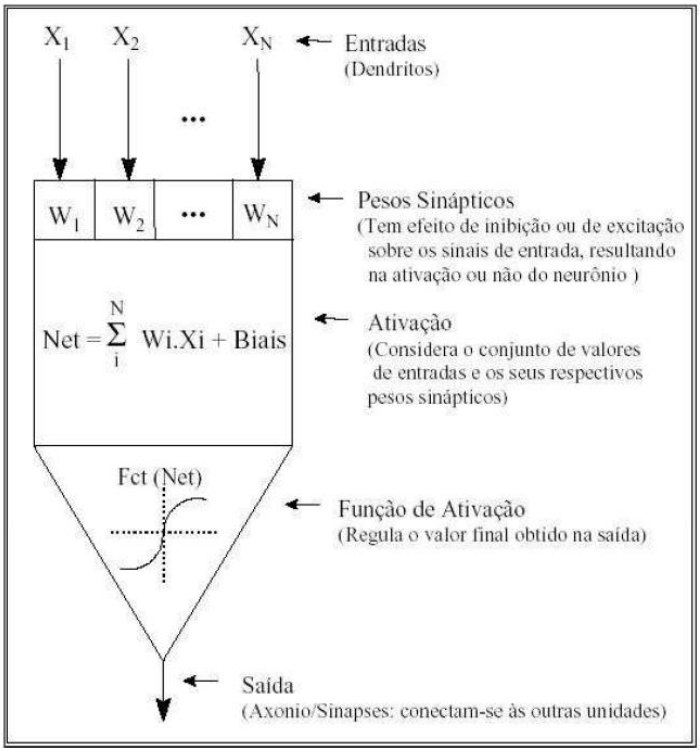


Figura 3.2: Neurônio artificial (OSORIO; BITTENCOURT, 2000).

Assim como o neurônio biológico, o neurônio artificial é constituído de: **a)** entradas, que recebem os sinais provenientes do exterior, **b)** uma função de ativação, que realiza o combinação dos valores obtidos nas entradas, e **c)** saídas, que transmitem os sinais recebidos e processados pelo neurônio para o exterior. Associado a cada entrada de um neurônio artificial existem pesos, que são valores numéricos que representam a força das conexões existentes entre os neurônios. Se um peo tiver um valor elevado, a sua respectiva entrada será amplificada, e se ele tiver um valor próximo de zero, a sua respectiva entrada terá seus valores atenuados (HAYKIN, 1999).

3.3 O *Perceptron* Elementar

Uma das primeiras iniciativas de modelar uma Rede Neural foi o *Perceptron*, que serviu de base para o desenvolvimento de algoritmos mais poderosos. A figura 3.3 ilustra o *Perceptron*, um neurônio artificial proposto por Frank Rosenblatt, inspirado no modelo *Psychon* de McCulloch e Pitts (HAYKIN, 1999). As percepções do ambiente chegam ao neurônio na forma de sinais de entrada *xi*, que são inibidos ou estimulados por um peso

sináptico *wi*. A soma ponderada de todos os sinais de entrada determinará o estado interno do neurônio, conhecido por *ativação interna*, denotado por *v*. Antes que o estado do neurônio seja propagado adiante, ele é aplicado a uma *função de ativação* (FA), também conhedia por *função de transferência* φ.

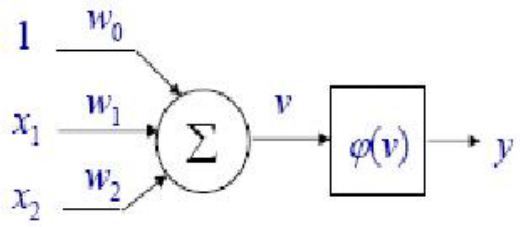


Figura 3.3: Modelo do neurônio artificial *Perceptron*.

A saída *y* de um neurônio *k* pode ser expressa então pela equação 3.1.

$$y_k = \varphi\left(\sum_{j=0}^{n} w_{kj}x_j\right) \tag{3.1}$$

onde *n* é o número de entradas do neurônio.

A entrada de valor 1 na figura 3.3 é uma entrada especial que não faz parte das percepções externas do neurônio. Esta entrada funciona como um limiar (*bias*) e normalmente tem seu valor fixando em 1 ou -1.

Como função de ativação, várias opções podem ser utilizadas pelos neurônios, estando algumas ilustradas na figura 3.4. A *função identidade* é uma função linear que simplesmente propaga o valor de ativação interna do neurônio, conforme demonstra a equação 3.2.

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

A *função limiar* binariza a saída do neurônio entre 0 e 1, segundo a equação 3.3.

$$f(x) = \begin{cases} 1 sex \geq 0; \\ osex \geq 0; \end{cases} \tag{3.3}$$

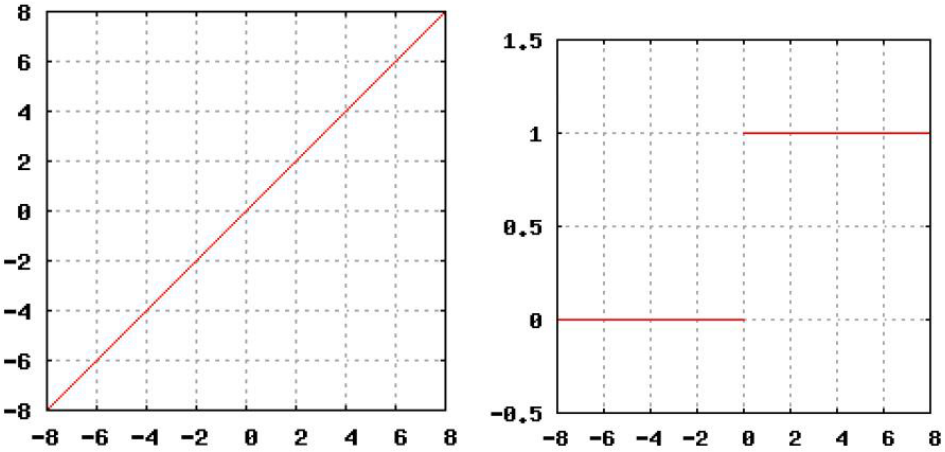
As figuras 3.4-c e 3.4-d representa a função sigmóide, conhecida por seu formato em "S". Essas duas funções, conhecidas respectivamente por *função logística* e *função tangente hiperbólica*, são comumente utilizadas em redes neurais que utilizam o algoritmo de treinamento *backpropagation*, pois tem uma derivada de baixo custo computacional.

A *função logística*, equação 3.4, limita os valores de saída dos neurônios entre 0 e 1, sendo sua derivada calculada pela equação 3.5.

$$g(x) = \frac{1}{1+\exp(-\sigma x)} \tag{3.4}$$

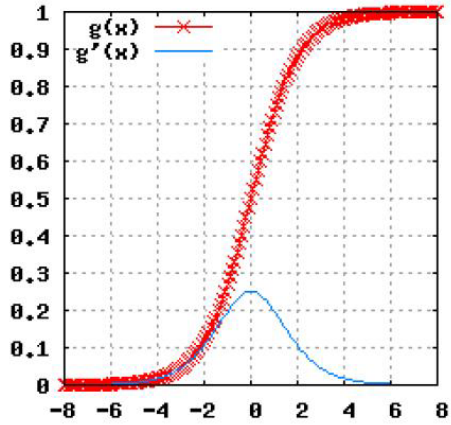
onde σ é o coeficiente de declividade.

$$g'(x) = \sigma g(x)[1-g(x)] \tag{3.5}$$

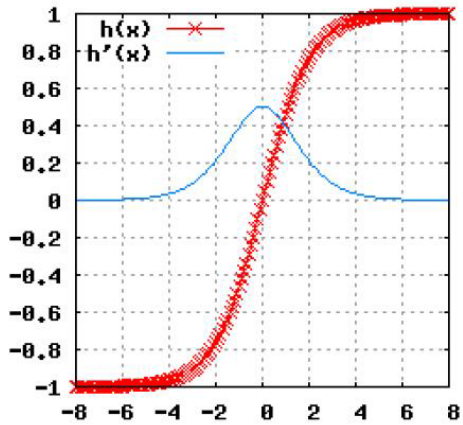


(a) Função Identidade.

(b) Função Limiar.



(c) Função Logística e sua derivada



(d) Função Tangente Hiperbólica e sua derivada

Figura 3.4: Funções de Ativação (HOFFMANN, 2006).

De forma semelhante, a *função tangente hiperbólica* (equação 3.6) limita as saídas dos neurônios entre -1 e 1, normalmente utilizada para dados de entrada binários (HAYKIN, 1999). Sua derivada é dada pela equação 3.7.

$$h(x) = \frac{1-\exp(-\sigma x)}{1+\exp(-\sigma x)} \tag{3.6}$$

$$h'(x) = \frac{\sigma}{2}(1-h^2) \tag{3.7}$$

Quando utilizadas de forma isolada, os neurônios não têm muito poder de processamento ou de armazenamento de informação. Todavia, várias arquiteturas de redes neurais artificiais foram propostas de redes neurais artificiais, de forma que quando combinados e conseqüentemente treinados, os neurônios possam resolver problemas complexos. Algumas dessas arquiteturas serão descritas nas próximas sub-seções.

O processo de treinamento das RNA é realizado através do ajuste dos pesos de cada unidade neural. No processo indutivo de aprendizado de máquina, as RNA podem ser treinadas de forma supervisionada ou não-supervisionada. O aprendizado supervisionado é realizado fornecendo-se dados de entrada para a RNA e as respectivas saídas que se deseja que a rede aprenda. Já os processos de aprendizado não-supervisionado, consistem

na sua maioria em técnicas de agrupamento, onde os dados de entrada são organizados em classes.

3.4 *Perceptrons* de Múltiplas Camadas

A rede *Perceptrons* de Múltiplas Camadas (MLP, do inglês *Multilayer Perceptrons*) consiste num arranjo de várias unidades neurais *Perceptron*, normalmente dispostas em camadas de ativação (HAYKIN, 1999). Uma rede MLP (figura 3.5) é formada por uma camada de entrada, nenhuma ou várias camadas ocultas e uma camada de saída. As unidades da camada de entrada servem como unidades sensoriais para coleta dos dados externos, não realizando nenhum tipo de processamento; apenas propagando o sinal para a próxima camada. O uso de uma ou mais camadas ocultas é opcional, mas necessário quando aplicado a problemas não linearmente separáveis. Já a camada de saída é a interface da rede que fornece os seus resultados ao mundo externo. O número de unidades nas camadas externas (entrada e saída) é diretamente dependente da dimensionalidade dos dados de entrada e saída da rede (HAYKIN, 1999). Quando o número de nodos e camadas estiverem adequados, as redes MLP funcionam como aproximadores universais, que podem realizar o mapeamento entre dois espaços de variáveis (HAYKIN, 1999).

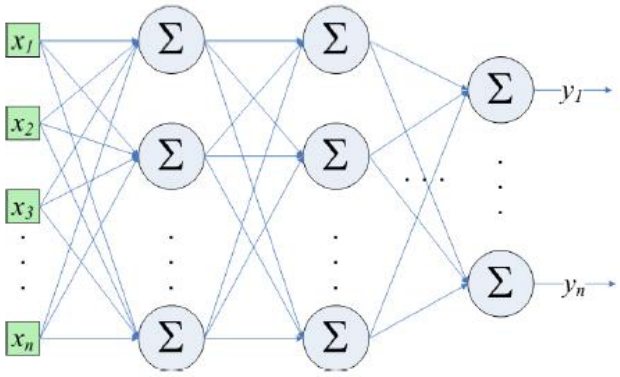


Figura 3.5: Modelo MLP.

A seguir, será descrito o algoritmo de *Backpropagation*, utilizado no processo de treinamento da rede MLP.

3.4.1 O Algortimo *Backpropagation*

O algoritmo de Retropropagação de Erros (*errorbackpropagation*) é uma das abordagens mais conhecidas para se efetuar o ajuste dos pesos de uma rede MLP. Trata-se de um processo de aprendizagem supervisionado, em que compara-se a saída obtida pela rede MLP com a saída desejada fornecida, calculando-se assim o Erro Quadrático Total da rede. O algoritmo *backpropagation* é um método de descida do gradiente que visa minimizar este erro, através do ajuste dos valores dos pesos sinápticos (HAYKIN, 1999). O ajuste do peso j do neurônio k é calculado conforme a equação 3.8:

$$w_{kj}(t+1) = w_{kj}(t) + \Delta w_{kj}(t) \tag{3.8}$$

onde w_{kj}(t) é o peso j do neurônio k no instante t.

O termo Δw_{kj} é a correção do peso, calculado por um gradiente local do neurônio e ponderado pelo valor de entrada da ligação sináptica e por uma taxa de aprendizado. É genericamente definido pela equação 3.9.

$$\Delta w_{kj}(t) = \alpha \delta_k(t) y_j(t) \tag{3.9}$$

onde: α é a tava de aprendizado;
$\delta_k(t)$ é o gradiente local, e
$y_j(t)$ é o valor de entrada da ligação sináptica j, ou seja, a saída do neurônio j da camada anterior à camada do neurônio k.

O gradiente local do neurônio k no instante t é dado pela equação 3.10.

$$\delta_k(t) = e_k(t) \varphi'_k(v_k(t)) \tag{3.10}$$

onde: e_k é o sinal de erro do neurônio k; e
φ' é a derivada da função de ativação (FA) do neurônio k.

Percebe-se então, que o sinal de erro $e_k(t)$, na equação 3.10, é um fator crítico para se efetuar o cálculo do ajust dos pesos do neurônio k. Determinar o sinal de erro na saída do neurônio k é uma tarefa que varia de acordo com a posição do neurônio na rede. Quando o neurônio pertence a camada de saída da rede, o sinal de erro é simplesmente calculado pela equação 3.11, ou seja, pela diferença entre o valor de saída do neurônio e o valor desejado $d_k(t)$.

$$e_k(t) = d_k(t) - y_k(t) \tag{3.11}$$

Para os demais neurônios, pertencentes a uma camada oculta da rede MLP, o sinal de erro e_k é calculado levando em conta o gradiente local dos neurônios da próxima camada e o peso de suas ligações sinápticas, conforme definido na equação 3.12.

$$e_k(t) = \sum_{l=1}^{m} \delta_l(t) w_{lk}(t) \tag{3.12}$$

onde: m é o número de conexões do neurônio k, ligando-se aos neurônios da próxima camada;
δ_l é o gradiente local do neurônio l, da próxima camada; e
w_{lk} é o peso de conexão entre os neurônios l e k.

Em (HAYKIN, 1999) encontra-se a derivação completa das equações do algortimo *backpropagation*.

Observa-se pela equação 3.10 que uma restrição importante do algoritmo *backpropagation* é que a FA dos neurônios deve ser uma função contínua e diferenciável (HAYKIN, 1999).

Em resumo, pode-se identificar três etapas no processo de treinamento:

a) Ativação: os dados de entrada $\bar{x}$ são propagados adiante na rede, até que se obtenha um vetor de saída $\bar{y}$;

b) Propagação do sinal de erro: a saída obtida pela rede é comparada com a saída desejada fornecida, calculando-se o sinal de erro da rede. Em seguida o sinal é propagado para trás pelas camadas da rede, enquanto se calcula o gradiente local de cada unidade neuronal;

c) Atualização dos pesos: o gradiente local de cada neurônio é utilizado para a correção de seus pesos sinápticos.

Essas três etapas são repetidas várias vezes para todo o conjunto de dados de treinamento, visando minimizar o Erro Médio Quadrático (MSE, do inglês *Mean Squared Error* da rede. O MSE é uma função de custo, representada pela equação 3.13, obtido pelo valor instantâneo $\varepsilon(n)$ da soma dos erros quadráticos de todos os neurônios da camada de saída da rede (equação 3.14). O número total de padrões utilizados no conjunto de treinamento é denotado por N, sendo n uma amostra deste conjunto.

$$MSE = \frac{1}{N}\sum_{n=1}^{N}\varepsilon(n) \tag{3.13}$$

$$\varepsilon(n) = \frac{1}{2}\sum_{k\in CS} e_k^2(n) \tag{3.14}$$

onde: CS é o conjunto de neurônios que pertencem a camada de saída da rede neural.

3.5 Redes Neurais Temporais

Diversas aplicações de redes neurais, como por exemplo reconhecimento de voz e processamento de sinais, apresentam um aspecto muito importante no processo de aprendizagem: o *tempo*. É a partir da introdução do tempo nestas operações que possibilitam obedecer variações estatísticas em processos não-estacionários, como os descritos anteriormente. Uma das perguntas então seria: Como colocar o tempo na operação de uma rede neural? Uma das maneiras para ser feito isto, segundo (HAYKIN, 1999), é a partir da *representação implícita*. Na representação implícita o tempo é representado pelo efeito que possui no processamento de sinaais numa maneira implícita. Por exemplo, o sinal de entrada é amostrado uniformemente, e a seqüência dos pesos sinápticos de cada neurônio conectado na camada de entrada da rede é convolucionado com uma seqüência diferente de amostradas de entrada. Assim, a estrutura temporal do sinal é embutida na estrutura espacial da rede.

Para um rede neural "estática" trabalhar com a variável tempo, é necessário adicionar propriedades *dinâmicas* (CLOUSE et al., 1997). Sendo assim, é necessário que a rede tenha *memória*. A memória pode ser classificada como sendo de **curta-duração** (*short-term*) ou **longa- duração** (*long-term*). Memórias de longa-duração são construídas através do aprendizado supervisionado, onde elas estão presentes nos pesos sinápticos da rede. Porém, para ter um comportamento dinâmico, a rede necessita de uma memória de curta-duração.

3.5.1 Memórias de curta-duração

O papel fundamental da memória é transformar uma rede estática em uma rede dinâmica (HAYKIN, 1999). Adicionando memória em uma estrutura estática de rede, como uma MLP, a saída da rede será uma função sobre tempo. Memórias de curta-duração podem ser implementadas em tempo contínuo ou discreto.

Uma prática ferramenta para tratar sistemas de tempo discreto é a *transformada Z*(GUPTA; JIN; HOMMA, 2003). Seja $x(n)$ uma seqüência discreta, a transformada Z, denotada por $X(z)$, é definida por:

$$X(z) = \sum_{n=-\infty}^{\infty} x(n)z^{-n} \tag{3.15}$$

onde z^{-n} é o *operador de atraso unitário*. Ou seja, z^{-n} funciona sobre $x(n)$ produzindo sua versão atrasada $x(n-1)$. Suponha que $x(n)$ seja aplicado em um sistema de tempo discreto de resposta de impulso $h(n)$, então a saída do sistema, dada por $y(n)$, é definida pela soma da convolução:

$$y(n) = \sum_{n=-\infty}^{\infty} h(k)x(n-k) \tag{3.16}$$

Os atributos de memória são medidos em termos de profundidade e resolução. Seja $g(n)$ a resposta completa de impulso da memória, definida por p sucessivas convuluções de $g(n)$, ou de forma equivalente, o inverso da transformada Z de $G^p(z)$. A profundidade de memória, denotada por D, é definida como o primeiro momento temporal:

$$D = \sum_{n=0}^{\infty} ng_p(n) \tag{3.17}$$

Uma memória com pequena profundidade D armazena informação para um intervalo relativamente pequeno de tempo, enquanto que uma memória com grande profundidade armazena informações do passado por mais tempo(HAYKIN, 1999). Resolução, denotado por R, é definida como o número de derivações de memória por unidade de tempo. Uma memória com alta resolução R é capaz de armazenar informações sobre a seqüência de entradas a um nível finito, enquanto que uma memória com baixa resolução pode somente armazenar informação a um pequeno nível (HAYKIN, 1999).

3.5.1.1 Memórias de atraso

Memórias de atraso consistem de p operadores de atraso unitário, cada um dos quais é caracterizado por $G(z) = z^{-1}$. Sendo assim, o *kernel* gerado é $g(n) = \delta(n-1)$, onde $\delta(n)$ é a unidade de impulso:

$$\delta(n) = \begin{cases} 1, & \text{n=0} \\ 0, & \text{n}\neq 0 \end{cases}$$

A figura 3.6 apresenta um diagrama da mais simples e largamente utilizada forma de memória de atraso de curta duração.

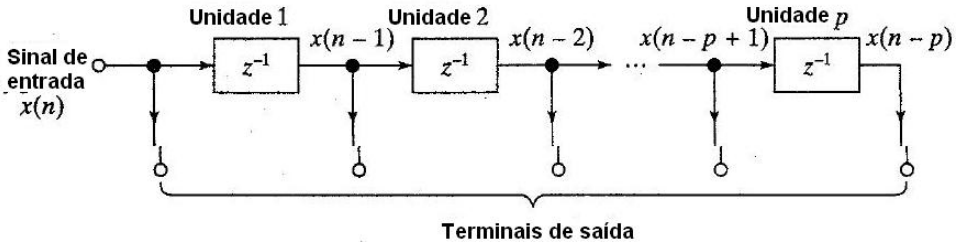


Figura 3.6: Memória de atraso de ordem p (HAYKIN, 1999).

O impulso de resposta da memória na figura 3.6 é $g_p(n) = \delta(n-p)$. Substituindo o $g_p(n)$ na equação 3.17 têm-se a profundidade de memória $D = p$. A partir da figura 3.6 é possível ver que existe apenas um atraso por unidade de tempo, então $R = 1$. Assim, a

profundidade da memória cresce linearmente com ordem p, e o produto de profundidade-resolução é contante em p (HAYKIN, 1999)

3.5.1.2 Memória Gamma

Este tipo de memória consiste de uma retro-alimentação com atraso unitário z^-1, em cada sessão, e um parâmetro ajustável μ que controla o comportamento da memória (HAYKIN, 1999). A função de transferência de cada sessão é dada por:

$$G_p(z) = \frac{\mu z^{-1}}{1-(1-\mu)z^{-1}} \tag{3.18}$$

A figura 3.7 apresenta o fluxo do sinal de um bloco $G(z)$ básico usado na estrutura da memória *Gamma*.

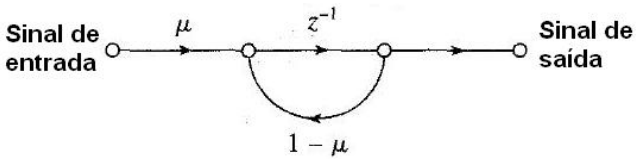


Figura 3.7: Fluxo de sinal de uma memória *gammap* (HAYKIN, 1999).

A profundidade de uma memória *Gamma* é p/μ e sua resolução é μ.

3.5.2 Redes *Feedforward* com Atraso Temporal Focado

Aplicações de reconhecimento de padrões temporais necessitam o processamento de padrões que desenvolvem-se no tempo, com a resposta em um determinado instante de tempo dependendo não apenas dos valores de entrada correntes, mas também de valores passados (HAYKIN, 1999). A figura 3.8 apresenta um filtro não linear construído sobre uma rede neural estática. A rede é estimulada através de uma memória de curta duração, onde o sinal de entrada é constituído pelo valor presente, $x(n)$, e p valores passados $x(n-1),...,x(n-p)$ armazenados em uma memória de atraso.

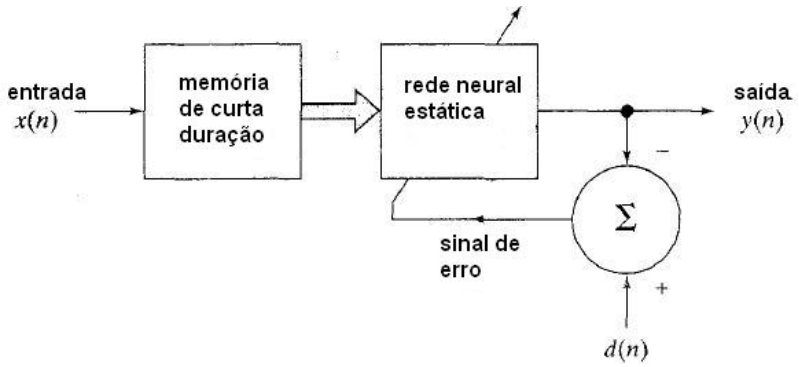


Figura 3.8: Filtro não linear sobre uma rede estática (HAYKIN, 1999).

A estrutura da figura 3.8 pode ser implementada ao nível de uma rede de neurônios (GUPTA; JIN; HOMMA, 2003). A figura 3.9 apresenta a rede chamada Rede *Feedforward* com Atraso Temporal Focado (do inglês *Focused Time Lagged Feedforward Network*, FTLFN).

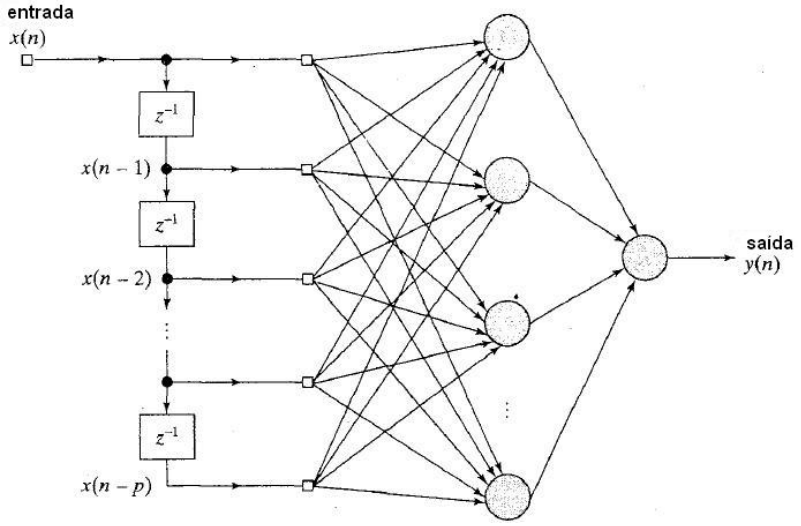


Figura 3.9: Rede *feedforward* com atraso temporal focado (HAYKIN, 1999).

A rede FTLFN da figura 3.9 possui como memória de curta duração a função z^{-1}, mas poderia ser perfeitamente alterada para usar a função $G(z)$. Esta arquitetura de rede é chamada *focada* porque a estrutura de memória está localizada completamente na camada de entrada. Para realizar o treinameto, é utilizado o algoritmo de *backpropagation* padrão, apresentado na seção 3.4.1. No tempo n, o padrão temporal aplicado para a camada de entrada da rede é o vetor $x(n) = [x(n), x(n-1), ..., x(n-p)]^T$. Uma época consiste em uma seqüência de padrões, ou seja, o número pelo qual é determinado pela ordem p da memória e o tamanho N dos exemplos de treinamento.

A saída da rede, assumindo que a rede MLP possua uma camada escondida, como mostrado na figura 3.9, é dada por

$$y(n) = \sum_{j=1}^{m_t} w_j y_j(n) = \sum_{j=1}^{m_1} w_j \varphi \left(\sum_{l=0}^{p} w_j(l) x(n-i) + b_j \right) + b_0 \tag{3.19}$$

onde a saída do neurônio é assumida como sendo linear e os pesos sinápticos do neurônio de saída são denotados pelo conjunto $\{w_j\}_{j=1}^{m_1}$, onde m_1 é o tamanho da camada oculta, e o bias é denotado por b_0

3.6 Aplicações de Redes Neurais

As RNAs podem ser aplicadas para a resolução de uma grande variedade de problemas. Devido a isso as pesquisas nesta área têm crescido consideravelmente. Suas características funcionais viabilizam sua utilização em casos onde há necessidade de manipulação de conhecimento impreciso ou ruidoso, além de possibilitar a construção de modelos a partir de exemplos.

Diversas aplicações envolvendo redes neurais têm sido desenvolvidas visando agregar vantagens a processos de tomada de decisão, em áreas como aplicações médicas, mercado financeiro, indústria química entre outros. Em geral, a principal aplicabilidade das redes neurais é encontrada em problemas que necessitam tratar o reconhecimento e classificação de padrões e aproximção de funções.

3.6.1 Reconhecimento de Padrões

A área de pesquisa que aborda o reconhecimento de padrões tem por objetivo a classificação de padrões em determinadas categorias ou classes. Compreende a técnica pela qual um sistema computacional é instruído através de exemplos a identificar determinados padrões, sendo então capaz de generalizar seu conhecimento e reconhecê-los novamente, mesmo que o padrão observado não seja exatamente igual ao que lhe foi apresentado como exemplo.

A escrita manual é um exemplo clássico de reconhecimento de padrões, pois cada pessoa possui um estilo próprio de escrever sendo, portanto, impossível mapear todos os estilos de grafia. No entanto, basta a apresentação de um pequeno conjunto de estilos de escrita para que através de comparações, seja possível identificar qualquer letra apresentada (CANSIAN, 1997).

O reconhecimento ou classificação de determinado padrão pode ser realizado de forma supervisionado, onde o padrão de entrada é identificado como um membro de uma categoria definida pelos padrões de treinamento, ou não supervisionado, onde o padrão é associado a uma categoria que é aprendida com base na similaridade entre os padrões apresentados durante o treinamento (LIMA, 2005).

3.6.2 Redes Neurais Aplicadas a Detecção de Intrusão

A aplicação de técnicas de inteligência artificial voltada à segurança computacional tem sido muito explorada nos últimos anos, objetivando otimizar os métodos convencionais de detecção.

Como todo conhecimento das redes neurais está localizado nos pesos das conexões sinápticas não é necessário realizar comparações exaustivas em tempo de produção entre bases de assinaturas e padrões de comportamento, proporcionando otimização dos recursos computacionais no processo de detecção de intrusão.

A maioria dos sistemas de detecção de intrusão existentes incorporam sistemas especialistas, que codificam seu conhecimento através de um conjunto finito de regras, sendo necessárias atualizações constantes para detecção de novos ataques, mesmo que estes sejam apenas variações de ataques já conhecidos.

O poder de generalização das redes neurais habilita a rede a reconhecer variações de um mesmo tipo de ataque e classificá-los de forma correta. Porém, com o surgimento de técnicas de intrusão completamente diferentes, faz-se necessário atualizar a base de treino e realizar o re-treinamento da rede neural, de forma a conferir o poder de adaptabilidade à rede. Algumas abordagens mais recentes visam otimizar esta situação, conforme detalhado em (MARKOU; SINGH, 2003).

Em (GEORGIA, 2000) são descritos processos de aprendizado que visam otimizar a adaptabilidade de sistemas de detecção de intrusão, possibilitando a detecção de novos ataques de forma autonôma. A incorporação de conhecimento é realizada por reforço contínuo, e não baseada na inserção de novas regras, como ocorre com os sistemas especialistas, nem no completo re-treinamento da rede como é o caso das abordagens tradicionais que utilizam redes neurais artificiais.

Diversas pesquisas têm sido realizadas de forma a explorar as características das redes neurais para a detecção de intrusão, como por exemplo:

- Em (RYAN; LIN; MIIKKULAINEN, 1998) é apresentado um sistema de detecção de intrusão denominado NNID (*Neural Network Intrusion Detection*), o qual utiliza uma rede neural para analisar a variedade de comandos executados pelos usuários,

e identificar alterações nos padrões de uso do sistema. Dependendo das características dessas variações, a utilização é considerada intrusiva.

- Em (JOO; HONG; HAN, 2003) é discutido o desempenho de sistemas de detecção de intrusão, sendo mensurado a partir de seus índices de falsos positivos e falsos negativos. Propõe ainda um modelo que utiliza redes neurais artificiais para aumentar a precisão da detecção realizada pelo sistema.

- A pesquisa apresentada em (DASGUPTA; BRIAN, 2001) descreve um modelo distribuído de detecção de intrusos, o qual aplica diversas alternativas computacionais em sua estrutura, como imunologia computacional, agentes móveis e redes neurais artificiais. O propósito dessa junção de tecnologias é criar um sistema robusto, eficaz e tolerante a falhas.

- A abordagem discutida em (CANNADY, 1998), leva em conta a construção de uma solução híbrida, que utiliza recursos de sistemas especialistas aliados às capacidades das redes neurais artificiais para otimizar o processo de detecção.

4 DESENVOLVIMENTO DO SISTEMA DE DETECÇÃO DE TRAPAÇAS

4.1 Introdução

O presente capítulo apresenta a descrição da solução proposta para o combate de *speed cheating* no jogo *Hoverkill*. São apresentados também o ambiente no qual este projeto está relacionado, mostrando a vulnerabilidade a trapaças em relação à outras arquiteturas de jogos, como foi abordado no capítulo 2.

Nas seções seguintes, é apresentado o jogo *Hoverkill*, o qual foi utilizado para ser testada a solução proposta. São abordados o protocolo do jogo, bem como sua dinâmica. Por fim, é feita uma descrição das abordagens de redes neurais artificiais utilizadas para a resolução do problema.

4.2 Visão Geral do Projeto P2PSE

O projeto P2PSE (desenvolvido no Instituto de Informática da UFRGS, e financiado pelo FINEP no período de 2006 - 2008), no qual este trabalho está inserido, propõe o desenvolvimento de um novo modelo de suporte a MMOG baseado nos paradigmas cliente-servidor e P2P de simulação. O modelo P2PSE combina modelos de suporte distribuído ou descentralizado a jogos multijogador e maciçamente multijogador que suportam uma baixa latência de comunicação entre os jogadores, de forma a suportar jogos de ação (como jogos FPS e de corrida) em um contexto maciçamente multijogador.

Em qualquer jogo cliente-servidor, maciçamente multijogador ou não, os jogadores interagem com o ambiente e com os outros jogadores através de um programa cliente. Cada cliente possui uma conexão de rede apenas com o servidor, como é mostrado na figura 4.1. O servidor é responsável por receber a informação de cada cliente e repassar atualizações para os outros clientes. Como os MMOGs são desenvolvidos para suportarem dezenas ou centenas de milhares de jogadores simultaneamente conectados ao sistema, isto frequentemente se traduz em um custo no lado servidor do sistema que não pode ser facilmente bancado por empresas de desenvolvimento de jogos de pequeno porte, bem como universidades ou centros de pesquisa que querem conduzir experimentos relacionados a MMOGs.

Neste contexto, o objetivo do projeto P2PSE é a redução de forma significativa do custo no "lado servidor"de um MMOG, com a motivação de tornar acessível o desenvolvimento e provimento de um MMOG por parte de uma empresa de pequeno porte ou mesmo de um grupo de pesquisa.No modelo proposto pelo P2PSE, o mundo virtual do jogo é sub-dividido em vários mundos, nos quais um determinado número de jogado-

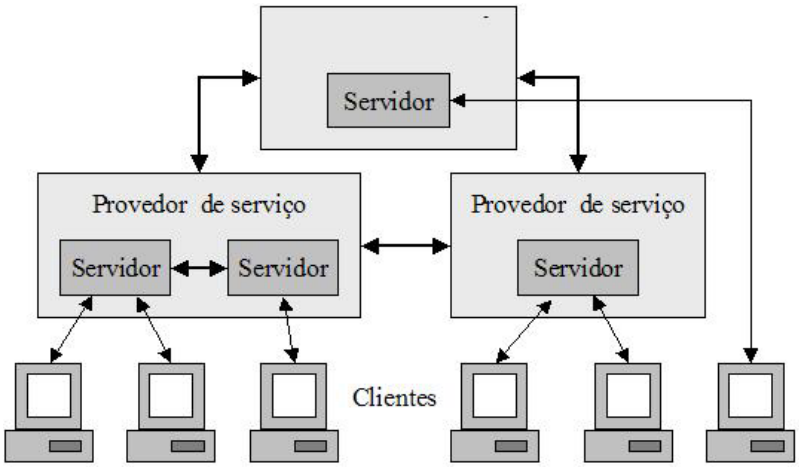


Figura 4.1: Arquitetura de jogo cliente-servidor.

res irá interagir. Esta interação é realizada apenas entre os pares de cada sub-divisão do mundo virtual, seguindo os conceitos do modelo de programação P2P, diminuindo consideravelmente o fluxo de dados na rede com o servidor, e aumentando a escalabilidade do jogo. A figura 4.2 ilustra como se dará essa nova interação.

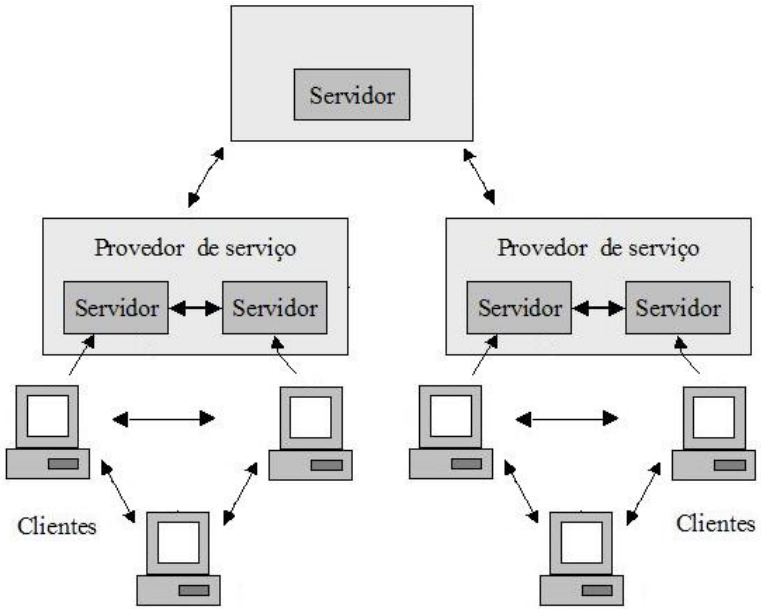


Figura 4.2: Modelo de arquitetura do P2PSE.

Como a comunicação com o servidor será apenas para a persistência do estado atual do jogo, a máquina a ser utilizada para este fim não necessita de alto poder de processamento, pois todo o trabalho de atualização do estado atual do jogo será feito por parte dos clientes.

Porém, a descentralização de um MMOG traz problemas de segurança, em especial, problemas com jogadores trapaceiros. Neste modelo, no qual não se faz presente um servidor que irá analisar cada uma das mensagens antes de repassar para os clientes, estes devem *confiar* na informação que estão recebendo dos demais jogadores, a fim de atualizarem o jogo em sua máquina para o estado atual.

4.2.1 Descrição do Jogo *Hoverkill*

O jogo no qual este sistema de detecção de fraudes foi inspirado é o *Hoverkill*, que foi desenvolvido pela empresa Singular Studios para a validação do modelo proposto pelo P2PSE. Este jogo é um MMOG de primeira pessoa, seguindo estilo FPS, onde existem determinados estilos de tarefas a serem realizadas, como por exemplo, capturar e defender a bandeira. A figura 4.3 apresenta um momento de ação do jogo.

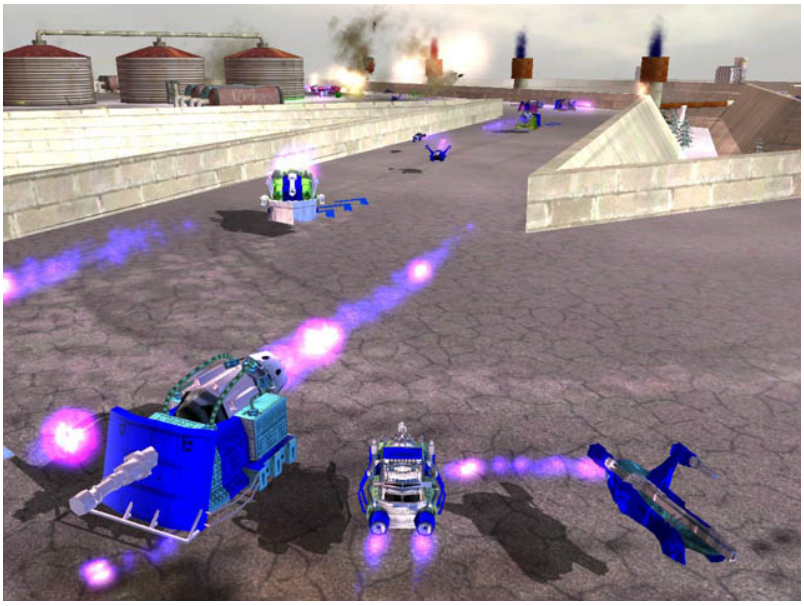


Figura 4.3: Tela do jogo *Hoverkill*.

Os personagens do jogo são definidos através de *hovers*, que são máquinas que flutuam no ambiente do jogo. Cada um destes *hovers* são inicializados com uma quantidade finita de energia, a qual é gasta à medida em que é realizado o deslocamento pelo cenário. Porém, esta energia é reposta com o passar do tempo e a diminuição do movimento.

Para a contagem de pontos do jogo, existem tarefas a serem feitas, como por exemplo um determinado grupo invadir uma certa região para a captura de uma bandeira *(flag capture)*, ganhando assim pontos. Outra forma de pontuação é alcançar determinados lugares dentro do mapa do jogo, onde existem *cubos* de energia, que rendem pontos para o *hover*. Essas são duas situações claras onde o uso de *speed cheating* poderia agregar enorme vantagem para um determinado jogador.

4.2.2 Protocolo do Jogo

O protocolo implementado pelo jogo *Hoverkill*, que é enviado de um cliente jogador para os demais, é apenas a informação referente à tecla que foi pressionada no teclado ou direção que foi informada no *joystick*. Essas informações, para que o jogo pareça em tempo real sem perturbações, são enviadas numa taxa de 20 mensagens por segundo.

Cada cliente após receber essas mensagens de direção dos demais clientes, irá transformar estas informações "brutas" em informações que façam sentido dentro do jogo. Para isto, existe um **motor de física**, o qual irá realizar essas transformações para informações de velocidades angulares e linerares de cada um dos *hoovers*, e a partir disto, são calculadas as posições atuais de cada um deles dentro do mundo virtual do jogo.

4.2.3 Definição da Categoria de Trapaça Detectada

Após levantamento das trapaças mais comuns em jogos *online*, apresentado na seção 2.5, foi detectado que a trapaça mais crítica em jogos no estilo do *Hoverkill* é o Aumento de Velocidade, conhecido por *Speed Cheating*. A criticidade desta ténica está no fato de permitir que um jogador chegue mais rápido em um ponto do jogo, seja para capturar algum bônus ou para aniquilar algum adversário mais facilmente. Em (CONSALVO, 2006), são apresentados os problemas causados por este tipo de trapaça em jogos mundialmente conhecidos.

Como já foi descrito, esta trapaça permite que um jogador aumente consideravelmente sua velocidade ou realize saltos dentro do mundo virtual do jogo. A figura 4.4 apresenta uma visão, na relação velocidade versus tempo, de como esta trapaça será entendida pelo motor de física do jogo.

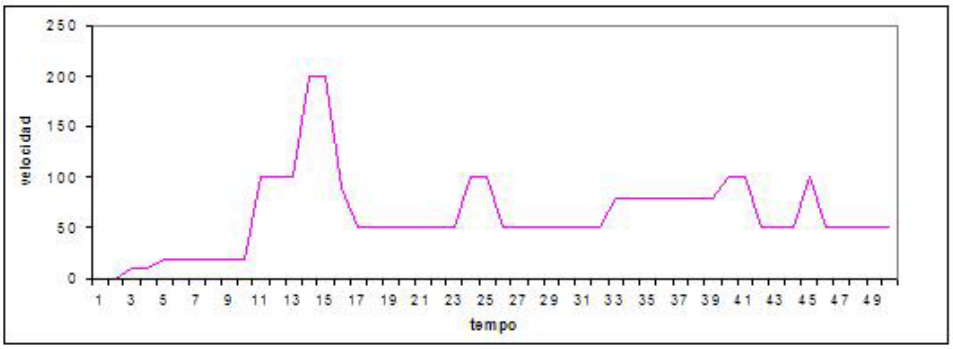


Figura 4.4: Gráfico de *speed cheating* em função da velocidade.

Já a figura 4.5 apresenta o comportamento de um jogador realizando os saltos no mundo virtual.

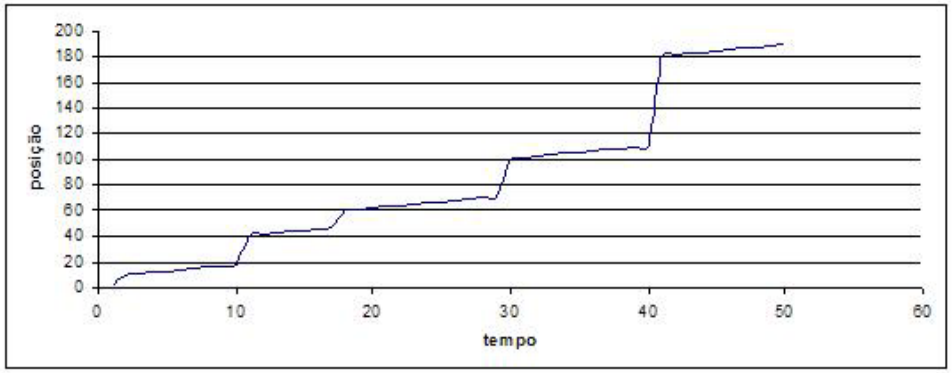


Figura 4.5: Gráfico de *speed cheating* em função da posição.

4.2.4 Problema dos Falsos-Positivos

Um dos problemas na detecção de *speed cheating* que foi levantado junto à equipe de desenvolvimento do jogo *Hoverkill*, é a ocorrência de **falsos-positivos**. Resumidamente, seria o fato de o módulo de detecção de fraudes proposto classificar um movimento honesto como sendo fraudoso. Este é um problema que deve ser evitado com alta prioridade, pois é mais aceitável não reconhecer uma fraude ao classificar um jogador como fraudoso, caso ele seja um bom jogador (CHAMBERS et al., 2005).

4.3 Modelo Computacional

O modelo proposto trata de um mecanismo de segurança capaz de detectar um comportamento considerado trapaceiro em uma simulação do jogo *Hoverkill*.

Esta detecção é realizada através de um módulo que possui características desejáveis baseadas em sistemas de detecção de intrusão, como a utilização de redes neurais para o reconhecimento do padrão de trapaças de forma a incorporar à estrutura habilidades, como adaptabilidade e generalização.

Estas capacidades possibilitariam a correta classificação de padrões semelhantes, mesmo que alguns destes padrões não façam parte do conjunto de treinamento. Essa característica colaboraria na composição de uma solução robusta e adaptável a mudanças no ambiente onde está inserida.

Apesar de parecer uma análise simples, onde somente comparações dos parâmetros com limiares bastariam para identificar anormalidades durante um jogo, pode se afirmar que isto não apresentaria resultados satisfatórios, visto que a simples comparação não levaria em conta os erros de comunicação, perdas de pacotes, o não-sincronismo do processo, entre outros fatores. Desta forma, fica evidente que é necessário fazer uso de alguma técnica computacional que permita analisar e identificar padrões e que tenham a capacidade de generalização, bem como ser resistente aos ruídos que possivelmente possam existir nos dados de entrada.

O modo de análise dos dados será feito de forma *online*, ou seja, em cada um dos clientes do jogo estará presente um módulo neural que irá analisar os movimentos enviados pelos demais jogadores. Esta alternativa leva vantagem em relação a um modo de análise *offline* pois não se faz necessário armazenar o *log* do jogo. Em compensação, têm-se uma perda de desempenho, mesmo que mínima, para a análise do modo neural. Porém, como este módulo não tem influência no decorrer do jogo, ele pode ser adicionado em uma *thread* separada do fluxo normal do P2PSE.

4.3.1 Problemas na Detecção de *Speed Cheating*

Como foi descrito em (CECIN; GEYER, 2006), uma solução óbvia para o combate de *speed cheating*, ao nível de aplicação, poderia ser adicionar uma comparação entre a posição reportada pelo nodo cliente com as posições informadas anteriormente pelo mesmo nodo, e declarando que se trata de uma trapaça quando o delta das posições excederem um determinado limiar. Esta solução é inflexível e introduz outros problemas. Primeiro, se existir um atraso considerável ou perdas de pacotes na rede, as mensagens podem chegar fora de ordem e causar falsos-positivos quando duas atualizações são enviadas em um intervalo de tempo distante pelo remetente e que chegam quase que simultaneamente ao nodo recebedor, parecendo assim que o nodo que enviou as mensagens está tentando fazer um *speed cheating*. Isto poderia ser contornado aplicando-se um rótulo de tempo (*timestamp*) para cada movimento do novo remetente, o que pode ser explorado pelos trapaceadores que modificam esta informação nos pacotes enquanto simulam estar com congestionamento no *link* da rede.
Segundo, esta abordagem depende em manualmente estimar o que é um movimento "ilegal", causando falsos-positivos e permitindo que trapaceiros usem de tentativa e erro , quais são os limiares e ajustar suas trapaças para explorarem este limite sem serem detectados. E finalmente, esta abordagem é inflexível e funciona para um tipo específico de trapaça e é amarrada à um protocolo de jogo.

4.3.2 Reação Passiva

O objetivo principal da solução proposta é detectar eventos considerados movimentos anormais dentro do jogo *Hoverkill*, e a partir deste reconhecimento apresentar os resultados probabilísticos aos administradores, como foi descrito na seção 2.7.4, que efetivamente poderão tomar ações reativas de acordo com a gravidade de cada caso específico, o que caracteriza um comportamento passivo por parte da solução.

4.3.3 Captura dos Dados para Treinamento

Os dados utilizados para o treinamento e teste do modelo proposto foram adquiridos através de várias rodadas do jogo *Hoverkill*. A partir do trabalho da equipe, estas informações foram coletadas de ambos jogadores honestos e trapaceiros, a fim de marcar no *log* em quais trechos do jogo houveram fraude ou não. Esse conjunto de dados é composto por 220 arquivos, cada um contendo em média 500 movimentos.

Para os dados referentes à trapaça, foram adicionados em alguns jogadores virtuais a característica de realizar "saltos" dentro do ambiente do jogo. Este saltos levam em consideração vários tipos possíveis de trapaceiros, tanto os que ficam intermitentemente realizando tais saltos quanto aqueles que utilizam com sutileza destas técnicas.

4.4 Módulo Neural

A seleção de uma rede neural artificial para ser aplicada ao domínio do problema deve ter por subsídio a avaliação de importantes critérios, tais como topologia, algoritmo de treinamento, taxa de aprendizado entre outros. Esses parâmetros devem ser ajustados de forma a melhor atender às necessidades do modelo proposto.

A fim de determinar qual a melhor arquitetura de RNA a ser aplicada, e dentro desta arquitetura quais os melhores parâmetros de configuração, foi utilizado o *toolbox* de redes neurais disponibilizado pelo ambiente *Matlab* (DEMUTH; BEALE; HAGAN, 2007). Dessa forma, a tarefa de comparação entre as arquiteturas de RNA se tornaria mais rápida, ao invés de se implementar as abordagens em alguma linguagem de programação. Após a validação do melhor modelo para resolução do problema proposto, será realizada a implementação do mesmo em linguagem C++, a fim do código ser facilmente integrado à biblioteca do P2PSE.

Quanto à arquitetura da RNA a ser utilizada, foi decidido fazer testes com as redes MLP e FTLFN. Estas arquiteturas foram selecionadas a partir dos estudos apresentados na seção 3.6, onde são mostrados trabalhos envolvendo classificação de comportamentos intrusivos. A escolha de se utilizar as redes FTLFN advieram do problema ter caracerísticas temporais associadas.

A partir das informações obtidas nos *logs* do jogo, as mesmas que são geradas pelo motor de física, o que será melhor abordado na seção 4.4.2, optou-se por utilizar apenas um neurônio na camada de saída da RNA, classificando o movimento em apenas duas categorias (trapaceiro e honesto).

De forma a evitar a ocorrência de falsos positivos ou falsos negativos a margem de saída estabelecida para caracterizar um movimento honesto ou trapaceiro pode ser configurada de acordo com o cenário. Como padrão, o sistema trata que o valor de saída deve estar no intervalo entre 0 e 1. Na saída da rede é aplicado um limiar. Caso o valor de saída seja maior que 0.95, o mesmo é considerado como sendo um movimento fraudulento. Foi decidido utilizar esta abordagem, para que um determinado movimento fosse considerado

como fraudulento apenas se a RNA tivesse "certeza" desta classificação, a partir do que foi aprendido durante a fase de treinamento.

4.4.1 Alternativas de Arquiteturas de Redes Neurais

Considerando que a tarefa consiste em classificação de padrões de comportamento a partir de exemplos, optou-se por utilizar as redes MLP e FTLFN. Baseado no levantamento feito das aplicações de reconhecimento de intrusão em redes de computadores, que são aplicações de reconhecimento de comportamento, e dos resultados obtidos, a utilização das redes MLP foram obrigatórias. Inicialmente, optou-se pelo uso de MLP, completamente conectado, com aprendizado supervisionado, a partir de um algoritmo de retropropagação de erros.

A justificativa da escolha das redes FTLFN deve-se ao fato do problema ter características temporais associadas. Esta rede, como foi analisada na seção 3.5, é uma variação das redes MLP utilizadas para análise e previsão de séries temporais. Por este fato somado a sua simplicidade de entendimento e implementação, esta rede será utilizada para a análise dos *logs* do jogo.

4.4.2 Dados de Treinamento

As informações que trafegam no jogo não consistem em dados prontos e interpretáveis, mas sim, de sinais de ativação de teclas. Dessa forma, qualquer que seja a opção de informação de entrada da RNA, essa deverá sofrer algum tipo de pré-processamento. A primeira opção sugerida foi trabalhar com as seguintes entradas:

- Velocidade Mínima
- Velocidade Máxima
- Velocidade Média
- Aceleração Mínima
- Aceleração Máxima
- Aceleração Média
- Deslocamento Mínimo
- Deslocamento Máximo
- Deslocamento Médio

No entanto, para chegar neste nível de detalhamento dos dados, são necessários vários cálculos além de que se estaria considerando um período onde se extraíria as médias e limites. Dessa forma, o sistema estaria perdendo eventos pontuais, onde as fraudes poderiam ocorrer.

A alternativa seguinte foi utilizar os dados num formato que o jogo já faz uso. A partir dos dados das teclas pressionadas, é efetuado um cálculo para identificar as velocidades angular e linear de cada eixo do veículo do jogador. Desta forma, utilizando somente estes dados, sem processamento estatístico, o problema maior se refere ao volume de dados, visto que cada jogador informa seus parâmetros a uma taxa de 20 vezes por segundo. Os novos parâmetros escolhidos são mostrados na tabela 4.1

Tabela 4.1: Dados disponíveis para o treinamento.

Variável	Tipo de dado
Velocidade Linear X	Numérico em ponto flutuante
Velocidade Linear Y	Numérico em ponto flutuante
Velocidade Linear Z	Numérico em ponto flutuante
Velocidade Angular X	Numérico em ponto flutuante
Velocidade Angular Y	Numérico em ponto flutuante
Velocidade Angular Z	Numérico em ponto flutuante
Uso do Turbo	Booleana
Energia Acumulada	Numérico inteiro

As velocidades lineares nos eixos X, Y e Z refletem a velocidade com a qual o *hover* se movimenta nos respectivos eixos. Já as velocidades angulares são relacionadas a velocidades com as quais o *hover* gira em torno do próprio eixo. Este movimento é utilizado quando é mirado um outro oponente. Os dados de energia acumulada se referem à quantia de energia que o *hover* possui para executar algum determinado movimento. Quanto ao uso do turbo, informando se o mesmo encontra-se habilitado ou desabilitado, refere-se ao fato do *hover* poder realizar movimentos com maior velocidade, respeitando a quantidade de energia acumulada no instante.

A partir deste conjunto de dados, pretende-se extrair o conhecimento necessário para reconhecer padrões relacionados ao *speed cheating*. Na fase de experimentos, será possível ter uma real noção de quais dados realmente serão extraídos os padrões, assim, possivelmente será reduzida a quantidade de variáveis que serão analisadas.

4.4.3 Treinamento

O treinamento tem como função agregar conhecimento à rede neural, atualizando os pesos das conexões sinápticas de acordo com as informações contidas nos dados de treinamento.

O algoritmo de treinamento utilizado no protótipo em questão é o *backpropagation*, melhor descrito na seção 3.4.1, onde cada época ou ciclo de aprendizagem representa uma apresentação completa de todo o conjunto de padrões. Após a apresentação de cada padrão é calculado o valor do erro da rede, que é retro-propagado atualizando os pesos sinápticos.

O protótipo implementado pode realizar o número necessário de ciclos até se atingir o nível de erro definido nas configurações do treino. Porém, é possível configurá-lo para que este passe a treinar a rede baseado em determinado número de ciclos, possibilitando um controle mais amplo sobre o número de épocas de treinamento.

O treinamento pode ser acompanhado por um gráfico, onde se pode analisar a curva de aprendizado gerada pela variação do erro à medida que os ciclos ocorrem, bem como observar o ponto de convergência do processo de treino. Ao se alcançar este ponto, dentro dos níveis aceitáveis de erro, a rede neural é considerada suficientemente treinada.

O treinamento se comporta de acordo com os parâmetros estabelecidos para taxa de aprendizado, *momentum* ou erro máximo. Para se definir os parâmetros ideais foram realizadas simulações com conjuntos de padrões durante a fase de testes.

Após o treinamento ser concluído é gerado um arquivo de pesos que representa todo o conhecimento adquirido. Estes pesos são atribuídos à rede neural tornando-a apta a realizar análises.

A geração deste arquivo de pesos representa uma forma de persistência do conhecimento adquirido, pois é possível a aquisição de conhecimento carregando os pesos deste arquivo para a rede neural, sem a necessidade de realizar novamente o mesmo treinamento. Desta forma é possível transferir todo conhecimento adquirido para outras redes com a mesma estrutura, bastando atualizar seus pesos de acordo com o arquivo estabelecido.

5 SIMULAÇÕES E RESULTADOS

5.1 Introdução

Este capítulo apresenta os experimentos e seus devidos resultados obtidos para a análise entre as duas arquiteturas de redes neurais, MLP e FTLFN. Essa atividade tem como objetivo principal reunir evidências que possam ser utilizadas para a validação da melhor arquitetura neural apresentada no capítulo anterior.

Os experimentos foram realizados para buscar a melhor arquitetura e configuração da rede neural, e também para detectar quais as variáveis que realmente influenciariam na detecção do *speed cheating*. A partir do conjunto de variáveis disponíveis para a análise, apresentadas no capítulo anterior, foram feitas podas nas mesmas, até se encontrar o conjunto que oferecesse os melhores resultados.

5.2 Treinamento

O treinamento da rede neural ocorre em ciclos ou épocas onde em cada época os padrões de treino são apresentados à rede neural e à cada apresentação é calculado o erro quadrático médio, através da diferença entre o resultado apresentado pela rede neural e o valor esperado para determinado padrão.

Os pesos das conexões sinápticas foram inicializados com valores aleatórios entre -1 e 1, os quais são ajustados no decorrer do treinamento.

No intuito de definir a melhor configuração para as redes neurais foram realizados experimentos variando o valor da taxa de aprendizado e a topologia da rede MLP, procurando-se observar o efeito desta variação sobre a convergência da rede. Os valores para a taxa de aprendizagem foram fixados em 0.5 e 0.8. Em relação à topologia, foram variados os parâmetros de entrada para a rede neural, a fim de descobrir quais variáveis iriam produzir melhores resultados para a classificação dos movimentos.

Para a rede FTLFN, os parâmetros de ajuste foram feitos em relação ao atraso que os dados de entrada deveriam sofrer. Foram considerados atrasos na ordem de $t-1, t-2$ e $t-3$.

O conjunto de treinamento para as redes foi definido em 800 movimentos, sendo que 400 eram referentes a movimentos honestos, e os outros 400 referentes a movimentos com fraude.

5.2.1 Configuração da Rede MLP

A partir dos dados de entrada apresentados na seção 4.4.2, foram realizados diversas simulações, onde foram variados a parâmetros da taxa de aprendizado, bem como as

variáveis à serem analisadas pela rede neural. Após sucessivas simulações, chegou-se na configuração ideal para a rede MLP.

Tabela 5.1: Configuração da rede MLP.

Algoritmo de Aprendizado: **Gradiente Decrescente com Momento**
Função de desempenho (erro): **Erro Médio Quadrático (MSE)**
Topologia: **7-4-1**
Função de ativação da camada intermediária: **Sigmóide Logística (LOGSIG)**
Função de ativação da camada intermediária: **Sigmóide Logística (LOGSIG)**
Ciclos de treinamento: **2000**
Taxa de aprendizagem: **0.5**
Momentum: **0.3**

A figura 5.1 mostra o gráfico do erro quadrático no treinamento da rede MLP. O erro máximo obtido foi de 0.04881.

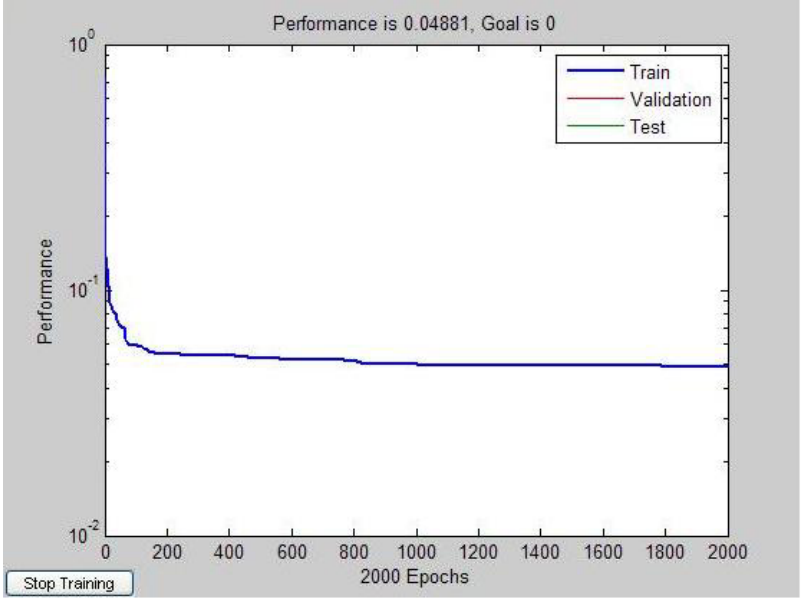


Figura 5.1: Gráfico do treinamento da rede MLP.

Quanto aos dados de entrada para a rede MLP, constatou-se que a **Energia Acumulada** representava um ponto de confusão na geração dos padrões, visto que ela poderia estar alta ou baixa, tanto em situações de fraude ou não. Portanto, chegou-se à conclusão de que para usar informação referente à energia, deveria não ser sobre o montante acumulado momentaneamente, mas sim, da energia gasta e recebida no período. No entanto, para isso, seriam então necessárias outras entradas de dados, como a localização de cubos de energia (que surgem no cenário de maneira aleatória), confirmação de que o jogador efetivamente alcançou um cubo e ganhou esta energia. Desta forma, considerando a dificuldade de uso dessa informação, optou-se por retirá-la da análise.

Teste realizados com outras configurações, mostraram um aumento no número de falsos positivos, causando assim perda na credibilidade do sistema de detecção.

5.2.2 Configuração da Rede FTLFN

Igualmente realizado para a rede MLP, foram executadas diversas simulações, variando-se também para esta arquitetura de rede o atraso da camada de entrada. A partir da configuração da rede MLP que apresentou os melhores resultados, foi tomada como base para as simulações com a rede FTLFN, porém, com o acréscimo da variável de **Energia Acumulada**. Como a rede FTLFN utiliza entradas atrasadas no tempo, esta variável que foi excluída na configuração da rede MLP, representou um índice a mais para a detecção de o movimento ser fraudulento ou não. A configuração que mostrou os melhores resultados foi a seguinte, apresentada na tabela5.2 :

Tabela 5.2: Configuração rede FTLFN.

Algoritmo de Aprendizado: **Gradiente Decrescente com Momento**
Função de desempenho (erro): **Erro Médio Quadrático (MSE)**
Topologia: **8-4-1**
Função de ativação da camada intermediária: **Sigmóide Logística (LOGSIG)**
Função de ativação da camada intermediária: **Sigmóide Logística (LOGSIG)**
Ciclos de treinamento: **100**
Taxa de aprendizagem: **0.5**
Momentum: **0.3**
Atrasos na entrada: **t-1, t-2**

A figura 5.2 apresenta o gráfico do erro na fase de treinamento. O valor do erro obtido foi de 0.0403232.

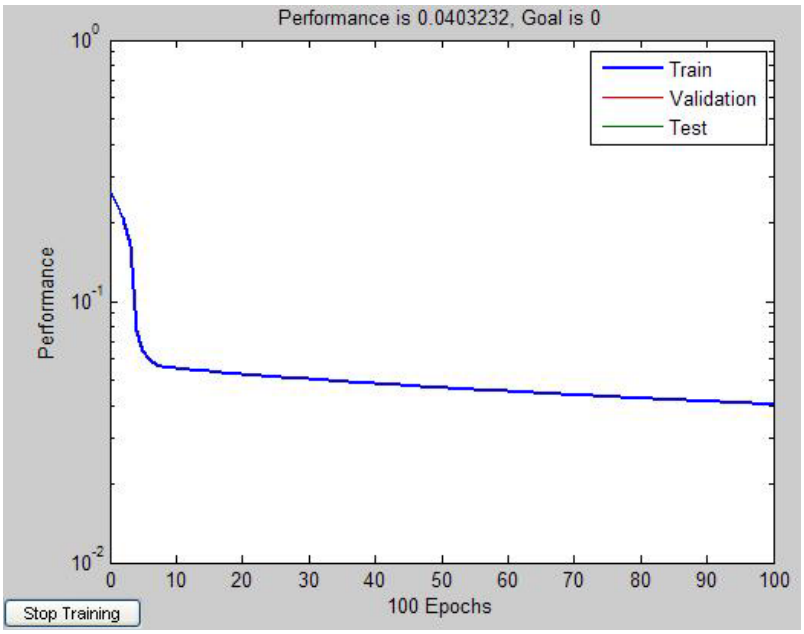


Figura 5.2: Gráfico do treinamento da rede FTLFN.

5.3 Validação dos Modelos

A resposta produzida pela rede neural para avaliação dos padrões de testes possui maior relevância ao mensurar o desempenho da rede, do que a apresentação dos próprios

padrões utilizados no treinamento. Dessa forma é possível avaliar a capacidade de generalização e adaptação do modelo baseado nas respostas geradas, obtendo-se resultados que retratam o comportamento da rede ao tratar casos não vistos durante a fase de aprendizado.

A fase de teste da rede, após treinada , foi executada sobre todo o conjunto de dados restante (excluindo os movimentos usados no treinamento) a fim de avaliar a generalização e aprendizado da rede neural. O conjunto completo é formado por 220 arquivos, contendo em média 500 movimentos. Isso equivale à 91 minutos de jogo, sendo referentes à 8 jogadores. A tabela 5.3 apresenta uma visão geral destes conjuntos de dados.

Tabela 5.3: Conjunto de dados para teste.

Jogador	**Qtd. Movimentos**	**Qtd. Fraudes**	**Percentagem Fraudes**
Player 1	11464	1349	11,76%
Player 2	13905	1862	13,39%
Player 3	13492	362	2,68%
Player 4	14633	8532	58,30%
Player 5	13372	1388	10,37%
Player 6	14602	219	1,49%
Player 7	14063	2027	14,41%
Player 8	13242	456	3,44%

5.3.1 Resultados

Após o devido treinamento, a rede neural é considerada apta a reconhecer e classificar corretamente qualquer padrão similar aos contidos na base de treino.

Para efetivamente testar a eficiência e a capacidade de generalização da estrutura, a rede neural foi devidamente treinada com o respectivo conjunto de treino, e os resultados da análise neural foram mensurados a partir do conjunto de testes, que foi formado por padrões não apresentados à rede neural durante o treinamento. Dessa forma os valores determinados pela rede neural podem ser considerados inferências realizadas a partir do processo de aprendizagem. A tabela 5.4 apresenta um resumo dos resultados obtidos com a rede MLP.

Tabela 5.4: Resultados do teste com rede MLP.

Jogador	**Qtd. Acertos**	**Qtd. Falsos-Positivos**
Player 1	514	4
Player 2	241	20
Player 3	231	363
Player 4	858	75
Player 5	807	209
Player 6	94	181
Player 7	350	42
Player 8	329	17

A figura 5.3 apresenta o gráfico de desempenho da rede, relacionando a quantidade de movimentos fraudulentos detectados com os movimentos de falsos-positivos, gerados pela rede neural, para cada jogador.

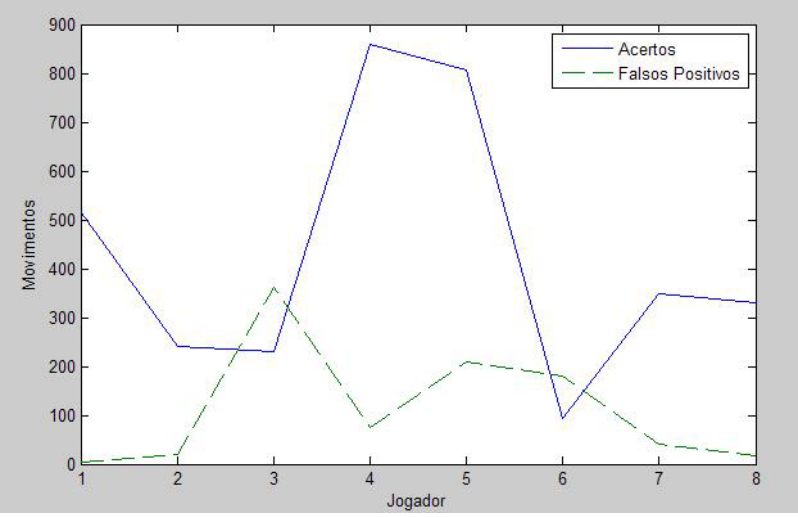


Figura 5.3: Gráfico de acertos e falsos-positivos do teste da rede MLP.

Por fim, foi realizado o teste com a rede FTLFN. Os resultados obtidos são apresentados na tabela 5.5.

Tabela 5.5: Resultados do teste com rede FTLFN.

Jogador	**Qtd. Acertos**	**Qtd. Falsos-Positivos**
Player 1	391	958
Player 2	403	202
Player 3	120	402
Player 4	5265	1558
Player 5	1010	209
Player 6	115	190
Player 7	1924	462
Player 8	420	99

A figura 5.4 apresenta o gráfico de desempenho da rede, relacionando a quantidade de movimentos fraudulentos detectados com os movimentos de falsos-positivos, gerados pela rede neural FTLFN, para cada jogador.

No intuito de melhorar a visão sobre o desempenho das redes em comparação com outras implementações de redes neurais, os acertos e falsos-positivos estão sendo apresentados na tabela 5.6 em uma escala percentual.

Tabela 5.6: Comparação de resultados entre rede MLP e rede FTLFN.

Acertos MLP	**Acertos FTLFN**	**Falsos-Positivos MLP**	**Falsos-Positivos FTLFN**
21,14%	59,57%	0,98%	4,40%

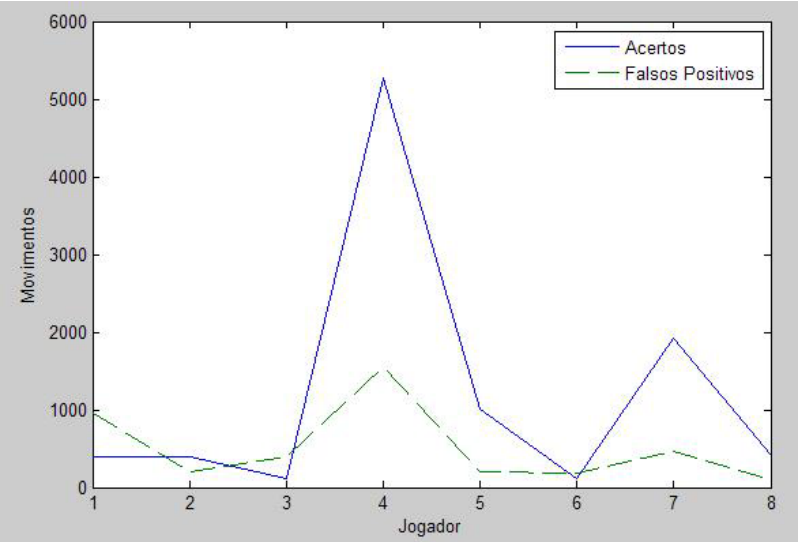


Figura 5.4: Gráfico de acertos e falsos-positivos do teste da rede FTLFN.

6 CONCLUSÕES

6.1 Introdução

Este capítulo faz uma série de considerações finais sobre o problema tratado nesta dissertação, as soluções propostas para o mesmo e os resultados obtidos para comprovação da viabilidade das soluções. São apresentadas também as conclusões e contribuições alcançadas a partir da realização da pesquisa e algumas sugestões de pesquisa futura decorrentes deste trabalho.

As funcionalidades de reconhecimento de padrões proporcionadas pelo uso de redes neurais artificiais permitiram a definição de um modelo simplificado de detecção de trapaças do tipo *speed cheating* em jogos *online* de computador.

A definição deste modelo e a respectiva implementação de um protótipo que atenda suas especificações foram contempladas neste trabalho. Esta pesquisa possibilitou mensurar resultados positivos, os quais contribuem para o uso de RNAs na parte de detecção de trapaças.

6.2 Conclusão Geral

A capacidade de generalização e adaptação obtida pela rede neural incorporada ao módulo de análise proporcionou vantagens significativas como a possibilidade de detectar com índices satisfatórios movimentos não previstos durante a fase de treinamento, conforme apresentado pelos experimentos realizados. Com os resultados de validação dos dois modelos de redes neurais, pode-se notar que a rede FTLFN apresentou um maior número de acertos, atingindo quase 60% de precisão, contra 21% da rede MLP. Porém, de acordo com o que foi proposto pelos desenvolvedores do jogo *Hoverkill*, o fato de prover um reconhecimento de fraudes sem gerar um grande número de falsos-positivos, a rede MLP leva vantagem em ter gerado menos de 1% do mesmo. A aquisição de conhecimento sobre novos comportamentos intrusivos ou normais pode ser facilmente obtida através da simulação e captura desse comportamento, incorporando-o ao conjunto de treino antes de realizar o re-treinamento da rede neural.

6.3 Trabalhos Futuros

Como sugestão para trabalhos futuros, podem ser citados:

- Utilização de outras arquiteturas de RNAs. Seria válido testar este tipo de problema com redes auto-organizáveis, como os Mapas de Kohonen; ou com redes convolucionais;

- Capturar dados referentes a outros tipos de trapaças, que foram apresentadas na seção 2.6, e incorporá-las aos dados atuais, a fim de obter-se um modelo para detecção de uma maior variedade de fraudes;
- Desenvolvimento de um classificador para ser incorporado ao administrador do jogo. Dessa forma, não se fará mais necessária a intervenção de um juiz (descrito em maiores na seção 2.7.4) para analisar as respostas que as RNA´s irão gerar;
- A reatividade do sistema é composta apenas por respostas passivas, sendo que em situações críticas haveria a necessidade que o sistema respondesse de forma ativa a eventos intrusivos.

6.4 Conclusão Final

Por meio deste trabalho pôde-se comprovar que técnicas de inteligência artificial podem contribuir efetivamente para a otimização de mecanismos aplicados à segurança como os sistemas de detecção de trapaças em jogos de computador.

Acredita-se que estas funcionalidades aliadas a outras técnicas existentes possam compor soluções mais eficientes e robustas no tratamento de movimentos fraudulentos em jogos, garantindo assim uma maior confiabilidade por parte tanto de servidores quanto dos usuários.

REFERÊNCIAS

BROMMELHOFF, A.; KING, J.; CLAYPOOL, M. **Improving Scalability in an Online Game**. 2005.

CANNADY, J. **Artificial Neural Networks to Misuse Detection**. 1998.

CANSIAN, A. M. **Desenvolvimento de um Sistema Adaptativo de Detecção de Intrusos em Redes de Computadores**. 1997. Tese (Doutorado em Ciência da Computação) — Universidade de São Paulo.

CECIN, F. R.; GEYER, C. F. R. P2PSE project: partially decentralized simulation for instanced mmogs. **WSPPD'2006**, Porto Alegre, RS, BR, 2006.

CHAMBERS, C.; FENG, W. chang; FENG, W. chi; SAHA, D. Mitigating information exposure to cheaters in real-time strategy games. In: NOSSDAV '05: PROCEEDINGS OF THE INTERNATIONAL WORKSHOP ON NETWORK AND OPERATING SYSTEMS SUPPORT FOR DIGITAL AUDIO AND VIDEO, 2005, New York, NY, USA. **Anais...** ACM Press, 2005. p.7–12.

CHEN, B. D.; MAHESWARAN, M. A cheat controlled protocol for centralized online multiplayer games. In: NETGAMES '04: PROCEEDINGS OF 3RD ACM SIGCOMM WORKSHOP ON NETWORK AND SYSTEM SUPPORT FOR GAMES, 2004, New York, NY, USA. **Anais...** ACM Press, 2004. p.139–143.

CHEN, Y.-C.; CHEN, P. S.; SONG, R.; KORBA, L. Online Gaming Crime and Security Issue - Cases and Countermeasures from Taiwan. In: PST, 2004. **Anais...** [S.l.: s.n.], 2004. p.131–136.

CLOUSE, D. S.; GILES, C. L.; HORNE, B. G.; COTTRELL, G. W. Time-Delay Neural Networks: representation and induction of finite-state machines. **IEEE Transactions on Neural Networks**, [S.l.], v.8, n.5, p.1065–1070, September 1997.

CONSALVO, M. **Gaining Advantage**: how videogame players define and negotiate cheating. [S.l.]: Ohio University, 2006. (889).

DASGUPTA, D.; BRIAN, H. Tmobile security agents for network traffic analysis. **DARPA Information Survivability Conference and Exposition**, [S.l.], v.01, n.3, p.1332, 2001.

DEMUTH, H.; BEALE, M.; HAGAN, M. **Neural Network Toolbox 5**: user´s guide. [S.l.]: The MathWorks, Inc., 2007. 849p.

GARFINKEL, T.; PFAFF, B.; CHOW, J.; ROSENBLUM, M.; BONEH, D. **Terra**: a virtual machine-based platform for trusted computing. 2003.

GAUTHIERDICKEY, C.; ZAPPALA, D.; LO, V.; MARR, J. **Low-Latency and Cheat-Proof Event Ordering for Distributed Games**. 2004.

GEORGIA, J. C. **Next Generation Intrusion Detection**: autonomous reinforcement learning of network attacks. 2000.

GUPTA, M. M.; JIN, L.; HOMMA, N. **Static and Dynamic Neural Networks**: from fundamentals to advanced theory. [S.l.]: John Wiley and Sons, 2003.

HAYKIN, S. **Neural networks**: a comprehensive foundation. [S.l.]: Prentice Hall, 1999.

HOFFMANN, L. T. **Aprendizagem por Reforço na Adaptação a Obstáculos em Navegação Robótica Autônoma Não-Estruturada Baseada em Imagens**. 2006. Dissertação (Mestrado em Ciência da Computação) — Instituto Nacional de Pesquisas Espaciais.

JAIN, A.; MAO, J. Artificial Neural Networks: a tutorial. In: IEEE JORUNALS, 1996. **Anais...** [S.l.: s.n.], 1996.

JOO, D.; HONG, T.; HAN, I. The neural network model for ids based on the asyimmetric costs of false negatives erros and false positives erros. **Expert System with applications**, [S.l.], v.01, n.3, p.69–75, 2003.

JR., J. L. B.; TORI, R.; JACOBER, E.; NAKAMURA, R. **A Survey on Networking for Massively Multiplayer Online Games**. 2004.

KABUS, P.; TERPSTRA, W. W.; CILIA, M.; BUCHMANN, A. P. Addressing cheating in distributed MMOGs. In: NETGAMES '05: PROCEEDINGS OF 4TH ACM SIGCOMM WORKSHOP ON NETWORK AND SYSTEM SUPPORT FOR GAMES, 2005, New York, NY, USA. **Anais...** ACM Press, 2005. p.1–6.

KIMPA, K. K.; BISSET, A. K. The Ethical Significance of Cheating in Online Computer Games. In: IRIE: INTERNACIONAL REVIEW OF INFORMATION ETHICS, 2005, Stuttgart, GE. **Anais...** ACM Press, 2005. p.263–268.

LI, K.; DING, S.; MCCREARY, D.; WEBB, S. Analysis of state exposure control to prevent cheating in online games. In: NOSSDAV '04: PROCEEDINGS OF THE 14TH INTERNATIONAL WORKSHOP ON NETWORK AND OPERATING SYSTEMS SUPPORT FOR DIGITAL AUDIO AND VIDEO, 2004, New York, NY, USA. **Anais...** ACM Press, 2004. p.140–145.

LIMA, I. V. M. de. **Uma Abordagem Simplificada de Detecção de Intrusão Baseada em Redes Neurais Artificiais**. 2005. Dissertação (Mestrado em Ciência da Computação) — Universidade Federal de Santa Catarina.

MARKOU, M.; SINGH, S. **Novelty detection**: a review-part 1: neural network based approaches. Disponivel em: <http://www.cs.ru.nl/ tomh/onderwijs/lrs/lrs files/review outliers.pdf>. Acesso em: 30 set. 2016.

OSORIO, F. S.; BITTENCOURT, J. R. Sistemas Inteligente baseados em Redes Neurais Artificias aplicados ao Processamento de Imagens. In: I WORKSHOP DE INTELIGÊNCIA ARTIFICIAL UNISC, 2000. **Anais...** [S.l.: s.n.], 2000.

RYAN, J.; LIN, M.-J.; MIIKKULAINEN, R. Intrusion Detection with Neural Networks. In: ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS, 1998. **Anais...** The MIT Press, 1998. v.10.

SHET, R. N. **Neural Network Based Analysis and Prediction of Deformation in Line Drawings**. 2004. Dissertação (Mestrado em Ciência da Computação) — Loughborough University.

SMED, J.; KAUKORANTA, T.; HAKONEN, H. Aspects of Networking in Multiplayer Computer Games. In: INTERNATIONAL CONFERENCE ON APPLICATION AND DEVELOPMENT OF COMPUTER GAMES IN THE 21ST CENTURY, 2001, Hong Kong SAR, China. **Proceedings...** [S.l.: s.n.], 2001. p.74–81.

SMITH, J. H. Trusting the avatar. **Article in Games and Culture**, [S.l.], v.5, n.3, p.298–313, 2004.

WEBB, S.; LI, K. **A Survey of Cheating Techniques in Online Games**. Disponível em: <https://home.cc.gatech.edu/webb/uploads/10/MiniProject3.pdf>. Acesso em: 30 set. 2016.

WITTENBURG, G. Modelling and classifying Cheating in Online Multiplayer Games. **Topics in Computer Systems Coursework. University of British Columbia**, [S.l.], 2004.

YAN, J.; RANDELL, B. A systematic classification of cheating in online games. In: NETGAMES '05: PROCEEDINGS OF 4TH ACM SIGCOMM WORKSHOP ON NETWORK AND SYSTEM SUPPORT FOR GAMES, 2005, New York, NY, USA. **Anais...** ACM Press, 2005. p.1–9.

YAN, J.; RANDELL, B. **Security in Computer Games**: from pong to online poker. School of Computing Science: Newcastle University, 2005. (889).

YAN, J.; RANDELL, B. A systematic classification of cheating in online games. In: NETGAMES '05: PROCEEDINGS OF 4TH ACM SIGCOMM WORKSHOP ON NETWORK AND SYSTEM SUPPORT FOR GAMES, 2005, New York, NY, USA. **Anais...** ACM Press, 2005. p.1–9.

ZETTERSTRöM, J. **A Legal Analysis of Cheating in Online Multiplayer Games**. 2005. Dissertação (Mestrado em Ciência da Computação) — Göteborg University.