PONTIFÍCIA UNIVERSIDADE DE SÃO PAULO - PUC-SP
PROGRAMA DE PÓS-GRADUAÇÃO EM TECNOLOGIAS DA INTELIGÊNCIA E DESIGN DIGITAL

MARCUS VINÍCIUS CARDADOR FRANCISCO

DESENVOLVIMENTO DE ALGORITMO PARA CONTROLE DE TRÁFEGO URBANO USANDO REDES NEURAIS E ALGORITMOS GENÉTICOS

SÃO PAULO -SP

2009

MARCUS VINÍCIUS CARDADOR FRANCISCO

DESENVOLVIMENTO DE ALGORITMO PARA CONTROLE DE TRÁFEGO URBANO USANDO REDES NEURAIS E ALGORITMOS GENÉTICOS

Dissertação apresentada à Pontifícia Universidade Católica de São Paulo como requisito à obtenção do título de Mestre em Tecnologias da Inteligência e Design Digital.

Orientador: Fernando Antônio de Castro Giorno.

Lista de Ilustrações

Lista de Tabelas

Lista de Siglas e Abreviaturas

AE	Ambiente Evolucionário
AG	Algoritmos Genéticos
EQM	Erro Quadrático Médio
FANN	Fast Artificial Neural Network Library
FIFO	First in first out
GNU GPL	GNU General Public License
MMIA	Média Móvel Integrada Auto-regressiva
MMPE	Média Móvel com Peso Exponencial
MPI	Message Passing Interface
RNA	Redes Neurais Artificiais
RNDT	Rede Neural por Defasagem de Tempo
SACI	Simple Agent Communication Infrastructure
SIMD	Single Instruction, Multiple Data
SMA	Sistema Multi Agente
VUA	Valor do Último Ano
VUM	Valor do Último Mês
XML	eXtensible Markup Language

Índice

1 Introdução

1.1 Motivação

A utilização de vias por veículos, pessoas ou animais, em grupos ou isolados, conduzidos ou não para fins de circulação, parada, estacionamento e operação de carga ou descarga é considerado trânsito segundo Lopes (1998).

Em especial nas grandes metrópoles, devido ao crescimento populacional de forma exponencial, as vias utilizadas para o trânsito tem se tornado incapazes para o escoamento de toda a demanda de veículos que circulam diariamente nesses centros.

Para facilitar no controle dessa demanda, fazem-se uso dos semáforos ou sinais de trânsito. Esses são compostos por três luzes em diferentes cores que são o vermelho, o amarelo e o verde onde cada uma representa uma ação a ser tomada pelo condutor do veículo quando deparado com o sinal. Em sua maioria, essas cores alternam-se de acordo com períodos de tempo previstos pelo administrador de tráfego embasando-se em alguns fatores, dentre eles na demanda de veículos prevista para uma determinada região em análise.

Uma tarefa de alta complexidade é definir quais os períodos de tempo mais adequados a serem usados em uma determinada região, levando-se em conta que fluxos de tráfego não são constantes e variam em diferentes horários e regiões tendo características peculiares e frequentemente mutantes. Para que haja a devida adequação dos períodos de tempo definidos nos atuais sistemas, geralmente há a necessidade da intervenção humana.

A presente da incerteza quando se tratando de transito, impõe dificuldades na implementação de soluções para a automação. Para o tratamento de problemas incertos, vem sendo desenvolvidas soluções baseadas

no funcionamento sináptico neural.

O funcionamento sináptico de neurônios biológicos inspirou o desenvolvimento de um neurônio artificial conhecido como *perceptron*. Os *perceptrons,* quando agrupados em camadas, compõem redes que tem a capacidade de identificar e classificar padrões. Problemas com possibilidades e soluções incertas, como ocorre no problema do transito, dificilmente obteriam uma resposta através de algoritmos convencionais. Enquanto que os padrões classificados por uma rede de *perceptrons,* baseada em um modelo matemático, poderia deduzir valores de ocorrências sucessoras ao momento presente.

Já os algoritmos de evolução genética ou evolucionários são geralmente utilizados para criar um ambiente cooperativo e/ou de concorrência que podem ou não incentivar os conhecimentos emergentes adquiridos pelas diferentes células que constituem esse ambiente. Nessa combinação, as células serão as Redes Neurais que irão interagir entre si buscando um incremento de seu conhecimento interagindo com outras células ao seu redor. Cada célula pode pertencer a uma rede, caso outra rede tente obter uma posição já ocupada haverá uma interação entre os dois elementos.

Partindo de uma metodologia híbrida fundamentada em algoritmos de propagação reversa e algoritmos evolucionários busca-se um aumento na eficiência do algoritmo de propagação reversa. Quando ambos os algoritmos são combinados, múltiplas redes interagem em um ambiente que pode ser cooperativo e/ou competitivo, tornando-se possível a diferenciação entre a qualidade dos resultados gerados pelas redes que, por sua vez, acarreta na possibilidade de seleção entre as redes que obtêm um maior nível de conhecimento.

O presente estudo direciona o enfoque no controle do tráfego de veículos em vias urbanas propondo a utilização de Redes Neurais para a análise de fluxos de tráfego e posterior tomada de decisão, afim de obter um escoamento relacionado à proporção do fluxo de trânsito. Essas redes tem como uma de

suas características a capacidade de interpretar alterações nos fluxos, podendo assim mudar as decisões de acordo com a demanda de tráfego.

1.2 Objetivo

Desenvolver um algoritmo híbrido combinando algoritmos genéticos e de propagação reversa para o controle do fluxo de tráfego que permita reduzir:

- Erros nas tomadas de decisão;
- O tempo de deslocamento de veículos, em especial nos horários de maior fluxo;
- A dependência presencial para interações com o sistema.

1.3 Método da Pesquisa

As seguintes atividades compõem o método da pesquisa utilizado:

- Estudo e apresentação de seminário sobre Redes Conexionistas, seus fundamentos teóricos e sua inspiração no modelo biológico;
- Estudo de modelos neurais de uma e múltiplas camadas;
- Estudo de diferentes algoritmos para implementação de Redes Neurais com análise de erros em apenas uma camada bem como em múltiplas camadas;
- Estudo sobre algoritmos de propagação reversa, percorrendo o trajeto histórico desde os primeiros desenvolvimentos dos neurônios artificiais;
- Análise de outras pesquisas direcionadas ao mesmo objeto de estudo (o controle do tráfego urbano);

- Desenvolvimento de algoritmo híbrido para o controle de tráfego;
- Definição de volume de trafego;
- Injeção dos dados nos algoritmo;
- Análise dos resultados e comparação com outros estudos;

1.4 Esquema Geral da Dissertação

Além da introdução (capítulo 1) a dissertação é composta por quatro capítulos.

No Capítulo 2 – Fundamentos Teóricos – é apresentado o fundamento das Redes Conexionistas e sua evolução bem como o fundamento de dois modelos de redes genéticas.

No Capítulo 3 – Estado da Arte – são apresentados estudos desenvolvidos com o intuito de analisar e controlar fluxos de tráfego. Esses estudos propõe diferentes soluções para o controle e predição viabilizando o desenvolvimento de um algoritmo híbrido com base nos melhores pontos de cada estudo.

No Capítulo 4 – Proposta para Controle do Tráfego Urbano – é abordado o projeto em questão, a descrição detalhada da combinação de como algoritmos de bases distintas trabalham de forma a desenvolverem uma arquitetura neural dinâmica em busca de um resultado próximo ao ideal, adaptando-se a constantes alterações nos fluxos de tráfego que ocorrem de forma intermitente.

No Capítulo 5 – Resultados – são apresentadas os resultados e as conclusões do estudo, uma comparação com estudos anteriores citados no Capítulo 3 e indicações para trabalhos futuros.

2 Fundamentos Teóricos

Esse capitulo apresenta os fundamentos teóricos essenciais para a contextualização da solução analisado no presente trabalho. Como parte de seu conteúdo estão a definição e estrutura de um neurônio artificial, técnicas utilizadas na aprendizagem dos *perceptrons* (neurônios artificiais), algoritmos para cálculos de erros e posterior propagação e definição de algoritmos que tratam problemas incertos baseados na evolução genética.

2.1 Fundamentos de Redes Conexionistas

Os primeiros estudos sobre os neurônios artificiais ocorreram de forma individual onde empregavam-se apenas um neurônio para resolver um problema. Esse neurônio recebia sinais de entrada e os tratava de tal forma a gerar um sinal de saída. Dessa forma problemas simples como as portas lógicas E e OU foram resolvidos, porém esses neurônios trabalhando de forma individual não são capazes de gerar soluções à problemas de maior complexidade como a porta lógica XOR ou OU exclusiva dado o fato de não haver uma separação linear nessa função. Após algum tempo, notou-se que por meio da combinação de dois ou mais neurônios os problemas não lineares obteriam uma resposta. Ao combinar dois ou mais neurônios tem-se uma Rede Neural ou conexionista.

2.1.1 O neurônio artificial

As arquiteturas conexionistas são frequentemente vistas como um desenvolvimento recente, porém é possível traçar suas origens a partir dos

primeiros trabalhos em Ciências da Computação, Psicologia e Filosofia. John von Neumann[1], por exemplo, foi fascinado por ambos, autômatos celular e modelos neurais computacionais. As pesquisas iniciais foram inspiradas em teorias psicológicas de aprendizagem de animais, especialmente as de Hebb (1949). Nessa seção, serão apresentados os componentes básicos à aprendizagem de uma Rede Neural e pontos históricos importantes nesta área.

O neurônio artificial, elemento base de uma Rede Neural, apresentado na **Figura 2.1**, é composto dos seguintes elementos:

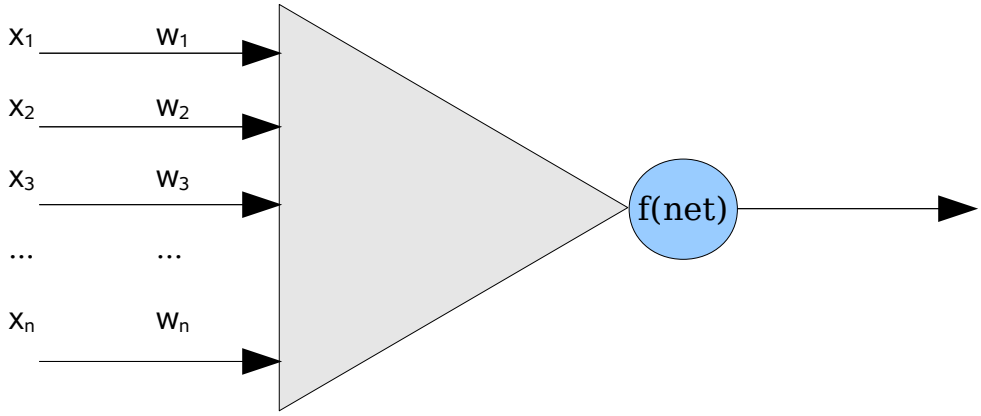

Figura 2.1 Um Neurônio Artificial (George F. Luger. 2002).

- *Sinal de entrada*, xi. Essa informação pode vir do ambiente ou da ativação de outros neurônios. Diferentes modelos variam nos valores de entrada permitidos; tipicamente o conjunto das entradas são valores discretos, do tipo {0,1} ou {-1,1}, ou ainda formado por números reais.

1 **John von Neumann,** nascido na Hungria em 28 de dezembro de 1903 com o nome de *János Lajos Neumann* e falecido em 8 de fevereiro de 1957 foi um matemático Húngaro-Americano que teve grandes contribuições em diferentes áreas incluindo Ciências da Computação, mecânica quântica, geometria continua, economia e teoria dos jogos, analise numérica, hidrodinâmica (de explosões), estatística e outros campos da matemática. Ficou conhecido como um dos maiores matemáticos do século XX (Wikipedia. 2009).

- *Um conjunto de pesos*, wi. Os pesos são valores reais e descrevem as forças das conexões.
- *Um nível de ativação*, Σ wixi. O nível de ativação de um neurônio é determinado pelo acumulo de forças dos sinais de entrada onde cada sinal é modificado pelo peso da conexão wi pertencente aquela linha de entrada. O nível de ativação é calculado pela soma das entradas multiplicadas a seu respectivo peso, isto é, Σ wixi.
- *Uma função de ativação*, **f**. Está função computa o neurônio final ou estado de saída, determinando o quão próximo de um valor delimitador encontra-se o nível de ativação do neurônio. A função de ativação é responsável por produzir o estado ligado ou desligado de cada neurônio.

Além das propriedades de cada neurônio acima apresentadas, também existem outras propriedades que caracterizam uma Rede Neural, como:

1. *A topologia de rede*. A topologia de uma rede é o padrão de conexões entre neurônios individuais. A topologia deve ser definida de forma a permitir o trabalho do algoritmo de aprendizagem. Podendo ser direta ou com realimentação, a topologia é a base para o bom funcionamento do algoritmo.
2. *O algoritmo de aprendizagem*.
3. *O esquema de codificação*. Isso inclui a interpretação da informação disponível no dado para a rede e o resultado de seu processamento.

2.1.2 Aprendizagem do perceptron

Frank Rosenblatt, em meados de 1958, desenvolveu um algoritmo para um tipo de rede com apenas uma camada, que recebeu o nome de *perceptron*.

Com relação às propagações de sinais, o *perceptron* era similar ao neurônio de McCulloch-Pitts (1943). Os valores de entrada e os níveis de ativação do *perceptron* são -1 ou 1; os pesos são valores reais. O nível de ativação do perceptron é a soma das multiplicações dos valores de entrada pelos seu respectivos pesos. O perceptron utiliza uma função para delimitar um valor de ativação que, quando atingido ou superado, gera uma saída de valor 1; caso contrario -1. Dados o valor de entrada $\mathbf{x_i}$, o peso $\mathbf{w_i}$ e o limite **t**, o perceptron calculará a saída da seguinte forma:

1 se $\Sigma\ \mathbf{x_i w_i} >= \mathbf{t}$

-1 se $\Sigma\ \mathbf{x_i w_i} < \mathbf{t}$

O perceptron baseia-se na forma de aprendizado conhecida como aprendizado supervisionado onde, depois de tentar resolver uma instância de um problema, o resultado esperado é fornecido e pode ser comparado com o resultado obtido. O *perceptron* então modifica os pesos na tentativa de reduzir o erro. A seguinte regra é usada: sendo **c** uma constante cujo valor determine o deslocamento do aprendizado e **d** o valor desejado na saída, o ajuste do peso na **i**ésima componente do vetor de entrada $\Delta\ \mathbf{w_i}$,é dado por:

$$\Delta\ \mathbf{w_i} = \mathbf{c(d - sign(}\ \Sigma\ \mathbf{x_i w_i))x_i}$$

O sinal, definido na expressão acima por $\mathbf{sign(}\ \Sigma\ \mathbf{x_i w_i)}$, é o valor de saída do perceptron. Isto é +1 ou -1. Então a diferença entre a saída desejada e a saída real será 0, 2 ou -2. Então, para cada componente do vetor de entrada:

Se a saída desejada é igual a real, não faça nada;

Se o valor de saída real é -1 e deveria ser +1, incremente os pesos na iésima linha executando $\mathbf{2cx_i}$;

Se o valor de saída real é +1 e deveria ser -1, decremente os pesos na **i**ésima linha executando $\mathbf{-2cx_i}$;

Esse algoritmo é capaz de gerar um conjunto de pesos que podem ser utilizados

para minimizar o erro médio no fim de todo o treinamento. Quando existir um conjunto de pesos capazes de gerar um valor de saída correto para cada um dos elementos de um conjunto de treinamento, então o algoritmo de aprendizagem do *perceptron* irá aprender isso (Minsky e Papert, 1969).

2.1.3 Redes Multi Camadas e o Algoritmo de Propagação Reversa

Os neurônios pertencentes a uma camada **n** em uma rede multi-camadas (veja **Figura 2.2**) se conectam às outras camadas ativando apenas os neurônios na camada **n + 1.** O processamento de sinais multi-camada indica que erros profundos na rede podem ser propagados por sucessivas camadas, dificultando assim a busca do(s) neurônio(s) causador(es) do(s) erro(s).

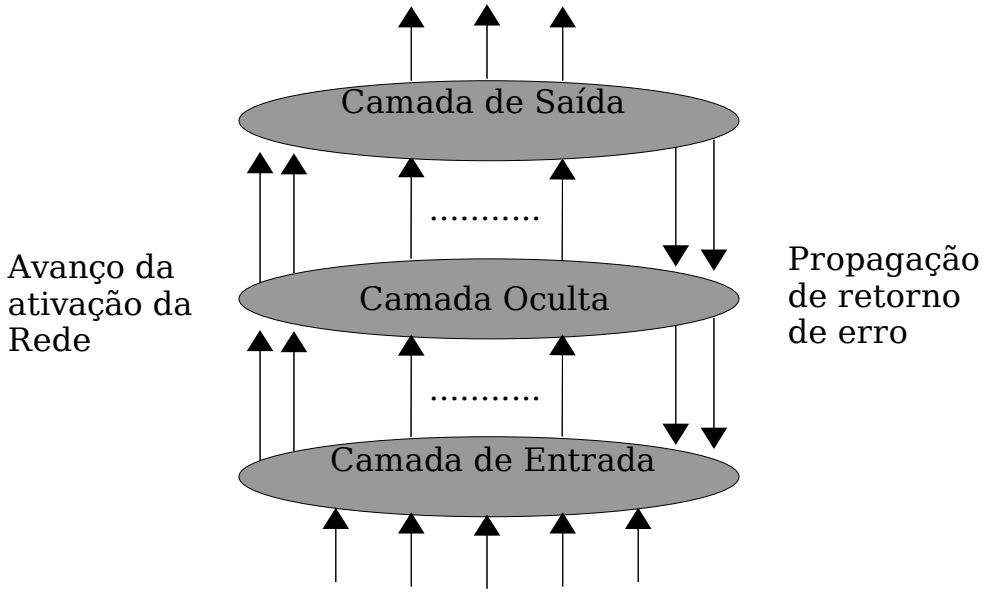

Figura 2.2 Propagação reversa em uma rede conexionista com uma camada oculta.

Dado esse fato, a análise da fonte de erro na camada de saída é complexa. A propagação reversa prove um algoritmo capaz de isolar o responsável e ajustar seus pesos de acordo com as necessidades.

A forma em que o algoritmo de propagação reversa trabalha é iniciando uma propagação dos erros pela camada de saída e enviando-os para trás, para a camada oculta.

Todas as informações necessárias para atualizar os pesos de um neurônio são locais para esse neurônio, exceto para o erro total. Para os nós de saída, esse é um calculo simples, como a diferença entre os valores de saída desejado e real. Para os nós das camadas ocultas, é consideravelmente complexo determinar o erro gerado por um nó especifico.

A função de ativação para a propagação reversa é geralmente a seguinte função lógica:

$$f(net) = 1/(1+e^{-\lambda * net}), onde\, net = \Sigma\, XiWi$$

Essa função é usada pelas seguintes razões: primeira, é uma função de forma senoidal; segunda, como é uma função continua ela é passível de ser derivada em qualquer ponto; terceira, como o valor da derivada é maior onde a função senoidal muda mais rapidamente a atribuição dos maiores erros é dada aos nós cuja ativação for a mais incerta. Finalmente, a derivada é calculada de uma forma simples, por subtração e multiplicação:

$$f'(net) = (1/(1+e^{-\lambda * net}))' = \lambda(f(net) * (1 - f(net)))$$

O treinamento na propagação reversa faz uso da regra delta generalizada (Georg F. Luger. 2002). Para os nós que estão na camada oculta, é vista a contribuição deles para os erros da camada de saída. As fórmulas para calcular os ajustes dos pesos **Wki** no caminho do **k**ésimo para o **i**ésimo nó no

treinamento de propagação reversa são:

1) $\Delta Wk = -c(di - Oi) * Oi * (1 - Oi) * Xk$

2) $\Delta Wk = -c * Oi(1 - Oi) * \Sigma(-DELTAj * Wij) Xk$

Em 2**), j** é o índice dos nós na próxima camada por onde os sinais **i** são injetados e:

$$DELTAj = -\frac{dErro}{dNETj} = (di - Oi) * Oi * (1 - Oi)$$

Seguem as derivadas dessas fórmulas. Inicialmente, a derivada de 1), a fórmula para o ajuste do peso nos nós da camada de saída. Como anteriormente, o objetivo é chegar a frequência de mudança de
erros da rede como uma função de mudança no **k**ésimo peso, Wk, do nó **i.**

$$\frac{dErro}{dWk} = -((di - Oi) f'(NETi) * Xk)$$

Como **f,** que poderia ser qualquer função, é agora a função lógica de ativação, então:

$$f'(net) = f'(1/(1 + e^{-\lambda * net})) = f(net) * (1 - f(net))$$

Recapitulando que $f(net_i)$ é simplesmente Oi, substituindo na equação anterior:

$$\frac{dErro}{dWk} = -(di - Oi) * Oi * (1 - Oi) * Xk$$

Como a minimização dos erros exige que os pesos mudem na direção do gradiente negativo, então será multiplicado por -c para ter um ajuste do peso para o **i**ésimo nó da camada de saída:

$$\Delta Wk = c(di - Oi) * Oi * (1 - Oi) * Xk$$

O próximo passo é a derivada do ajuste do peso para a camada oculta. Para que fique claro, inicialmente será assumido somente uma camada oculta. Analisando um único nó**, i**, da camada oculta, será identificado sua contribuição para o total de erros da rede. Isso é feito a partir de uma análise inicial das contribuições de erros dos nós **i** em um nó **j** na camada de saída. Então essas contribuições são somadas por todos os nós da camada de saída. Finalmente, descreve-se as contribuições dos **k**ésimos pesos de entrada no nó **i,** para o erro da rede. A **Figura 2.3** ilustra essa situação.

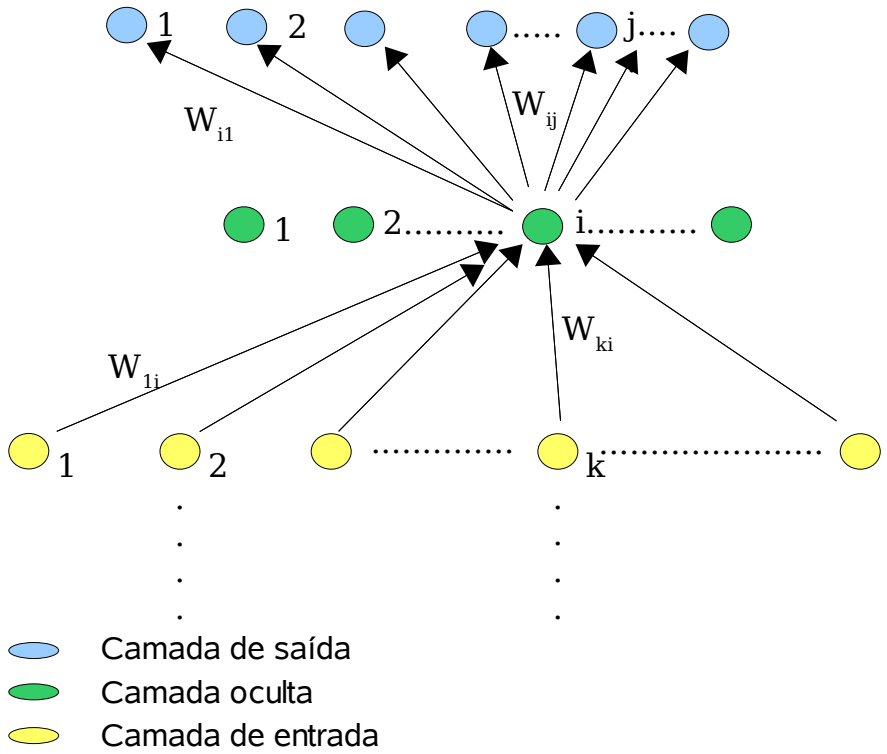

Figura 2.3 Contribuição total do nó i para o erro na saída.

Inicialmente, realiza-se a derivada parcial do erro da rede considerando a saída do nó **i** na camada oculta:

$$\frac{dErro}{dOi} = \frac{dErro}{dNETj} * \frac{dNETj}{dOi}$$

O negativo do primeiro termo do lado direito da igualdade, é chamado de DELTAj. Então pode-se escrever:

$$\frac{dErro}{dOi} = -DELTAj * \frac{dNETj}{dOi}$$

Recordando que a ativação do nó **j, NETj,** na camada de saída é dada pela somatória do produto dos pesos com os valores de saída dos nós da camada oculta:

$$NETj = \Sigma Wij * Oi$$

Executando a derivada parcial considerando apenas um componente da somatória, nomeado a conexão entre o nó **i** e o nó **j:**

$$\frac{dNETj}{dOi} = Wij$$

onde **Wij** é o peso na conexão do nó **i** na camada oculta para o nó **j** na camada de saída. Substituindo o resultado:

$$\frac{dErro}{dOi} = -DELTAj * Wij$$

Agora é feita a soma de todas as conexões do nó **i** na camada de saída:

$$\frac{dErro}{dOi} = \Sigma - DELTAj * Wij$$

Isso dá a sensibilidade do erro da rede na saída do nó **i** na camada oculta. Em seguida, será determinado o valor de DELTAi, a sensibilidade do erro da rede para a ativação da rede no nó oculto **i.** Isso dá a sensibilidade do erro de rede para os pesos de entrada do nó **i:**

$$-DELTAi = \frac{dErro}{dNETi} = \frac{dErro}{dOi} * \frac{dOi}{dNETi}$$

Como está sendo usada a função lógica de ativação,

$$\frac{dOi}{dNETi} = Oi * (1 - Oi)$$

Agora este valor é substituído na equação DELTAi:

$$-DELTAi = Oi * (1 - Oi) * \Sigma - DELTAj * Wij$$

Finalmente, poderá ser verificada a sensibilidade de erro da rede na camada de saída para os pesos de entrada no nó oculto **i.** Examinando o **k**ésimo peso no nó **i, Wk.** Por meio da seguinte regra**:**

$$\frac{dErro}{dWki} = \frac{dErro}{dNETi} * \frac{dNETi}{dWki} = -DELTAi \frac{dNETi}{dWki} = -DELTAi * Xk$$

onde **Xk** é a **k**ésima entrada do nó **i.**

Substituindo na equação o valor de -DELTAi:

$$\frac{dErro}{dWki} = Oi(1-Oi)\Sigma(-DELTAj * Wij)Xk$$

Como a minimização dos erros exige que os pesos mudem na direção do gradiente negativo, o ajuste do peso é feito, para o **k**ésimo peso de i, pela multiplicação pela constante de aprendizagem negativa:

$$\Delta Wki = -c\frac{dErro}{dWki} = c * Oi(1-Oi)\Sigma(DELTAj * Wij)Xk$$

Para redes com mais de uma camada oculta, o mesmo procedimento é aplicável recursivamente para propagar o erro da camada oculta **n** para a camada oculta **n – 1.**

Embora produza uma solução para o problema de aprendizagem em redes de múltiplas camadas, a propagação reversa apresenta algumas dificuldades próprias. Como em uma escalada, ela pode convergir para um ponto de mínimo. Finalmente, propagação reversa pode ser de difícil execução, especialmente quando a rede converge lentamente.

2.2 Algoritmos de Evolução Genética

Os algoritmos de evolução genética são algoritmos que tomam como fonte de inspiração a evolução de cromossomos. Esses algoritmos podem aparecer em diferentes configurações baseadas em etapas do cruzamento genético. O presente estudo tem foco em duas configurações que são conhecidas como algoritmos de População Caótica e algoritmos de Emulação Parcial.

2.2.1 População Caótica

Esse algoritmo é inspirado no fato conhecido de que em ambientes naturais, tamanhos populacionais, reprodução e níveis sociais estão em constante mudança tendendo a estabilizar-se próximos a valores de acordo com alguns fatores do ambiente (Annunziato e Pizzuti, 2002). Dessa forma, foi desenvolvida uma técnica para definir parâmetros genéticos durante um curso de desenvolvimento, adaptando-se o tamanho populacional e operadores de acordo com um controle ambiental e densidade populacional. O algoritmo de população caótica prevê encontros, reprodução monossexual, reprodução bissexual e competição dos indivíduos.

Nesse modelo, para cada iteração é selecionado um indivíduo, o **i**ésimo da população sendo i entre 1 e o tamanho atual da população; somente então inicia-se a busca por um segundo indivíduo. Porém, a probabilidade do encontro é definida de acordo com a densidade populacional. Nesse estágio, caso haja um encontro, então uma iteração (reprodução bissexual ou competição) irá começar; caso contrario uma reprodução monossexual do indivíduo pode ocorrer.

A reprodução bissexual ocorre de acordo com uma taxa adaptativa e, se ocorrer, os filhos resultados da relação não substituem os pais; pelo contrario, são simplesmente adicionados à população. Nesse caso, a população aumenta em dois elementos.

A reprodução monossexual também depende da taxa adaptativa e, quando ocorre, um indivíduo inicialmente se clona e em seguida passa por um processo de mutação. O indivíduo que sofre a mutação não substitui o original, aumentando o tamanho populacional em uma unidade.

A competição, não diferente das demais, também é dependente da taxa adaptativa. No caso de dois indivíduos se encontrarem e houver a impossibilidade da reprodução bissexual então haverá uma competição; nessa,

os indivíduos irão lutar pela sobrevivência e o indivíduo mais forte irá expulsar o mais fraco, eliminando-o da população.

2.2.2 Emulação Parcial

Sob um plano bidimensional, múltiplos indivíduos se movem pelas células disponíveis, podendo essas serem ocupadas por um único individuo por vez. Na tentativa de um indivíduo entrar em uma célula já ocupada, haverá uma iteração.

Todos os parâmetros para dinâmica, reprodução, vida e iterações são armazenados em um mapa genético definido durante o nascimento de cada indivíduo e permanecem imutáveis durante toda a sua vida.

Durante suas vidas, os indivíduos podem se mover entre as células disponíveis; porém, a dinamicidade e a probabilidade de iterações dependem de outras variáveis como o número de agentes, as modalidades de iterações e o tamanho do espaço disponível.

Esse algoritmo é inspirado em uma sociedade cooperativa onde otimizações são obtidas por meio de mecanismos de emulação de comunicações.

As metáforas da evolução e genética não são aplicáveis nesse modelo pelo fato da população ter um tamanho fixo e os indivíduos não morrerem ou se reproduzirem.

A otimização consiste em uma ordenação da adaptação no desenvolvimento. No início, um número fixo de indivíduos é colocado ao redor e começam a se movimentar. Cada um deles é inicializado com um bônus de energia. Parte dessa energia é paga pelo indivíduo a cada novo ciclo de vida. No encontro de dois indivíduos, um mecanismo de competição é ativado. A competição é baseada no valor de resistência de cada indivíduo. O indivíduo

mais fraco transfere parte de sua energia para o indivíduo mais forte.

Ao mesmo tempo ocorre uma reação contrastante cooperativa onde o indivíduo mais fraco adquire parte das habilidades do indivíduo mais forte possibilitando assim uma emulação parcial das habilidades do mais forte pelo mais fraco. Essa emulação parcial consiste em uma alteração no peso do indivíduo de acordo com a seguinte fórmula:

$$W_{li} = a*W_{wi} + (1-a)*W_{li}$$

Onde W_{wi} representa o peso genérico do indivíduo com menores habilidades, W_{li} é o peso genético do indivíduo mais hábil, *a* representa o fator de emulação fixado em 0,05. Sempre que um indivíduo chega a um nível de energia muito baixo ou próximo de zero, esse tem o seu bônus reinicializado pela rede. A contínua reinicialização é importante inclusive para manter a biodiversidade do ambiente.

Boa parte dos indivíduos reinicializados voltam a perder sua energia por não terem as habilidades necessárias para ganhar mais energia; porém, parte deles é bem sucedida e tem a possibilidade de mudar para uma classe mais desenvolvida.

Um contraste novamente ocorre na situação onde o indivíduo mais forte, ao absorver energia de um indivíduo mais fraco, transfere parte de suas habilidades podendo transformá-lo em um indivíduo mais hábil que, por sua vez, será capaz de tomar energia do indivíduo até então com maior resistência. Isto atenuará a resistência do indivíduo mais forte podendo torná-lo tão fraco a ponto de ser reinicializado. Isso ocorre quando sua resistência aproxima-se de zero.

Esse algorítimo difere do anterior na forma de aprendizado por ser baseado em um desenvolvimento coletivo e de alta volatilidade. O conhecimento aqui é um produto de toda a sociedade, gerado durante a vida dos indivíduos e que se move dinamicamente entre eles, podendo ser transferido entre gerações.

Um dos pontos mais interessantes nesse algoritmo é o balanço entre a competição e a cooperação, zelando pelo conhecimento da sociedade e pela formação de nichos de evolução. Também é um bom mecanismo para promover novas soluções e proteger as emergentes.

3 Estado da Arte

Nesse capitulo são apresentados estudos sobre alternativas para o controle e predição do tráfego urbano. Esses estudos estão fundamentados em algumas vertentes das Redes Neurais artificiais, que serão detalhadas ao desenvolver do capitulo, e buscam soluções para tratar a demanda de tráfego em diferentes situações.

3.1 Previsão de tráfego em vias urbanas por meio de Redes Neurais geneticamente desenvolvidas (Lingras et al. 2006)

3.1.1 Descrição

Dentre os estudos desenvolvidos para predição de trânsito em vias urbanas, grande parte está focada em uma predição a curto prazo. Essas pesquisas geralmente apresentam duas limitações. A primeira é que na maioria delas a previsão é baseada em apenas um intervalo de tempo (ex.: 1min, 1hora). A segunda é a dificuldade encontrada para tratar situações especiais como acidentes, manutenções ou congestionamento em vias. Esses tipos de incidentes podem ter uma duração variável, fazendo-se muitas vezes necessário a utilização de um sistema que possa suportar múltiplos intervalos de tempo.

Esse estudo (Lingras et al. 2006) propõe a utilização de diferentes modelos como os modelos naïve, análise de series de tempo e Redes Neurais geneticamente desenvolvidas para a previsão de 12 horas do volume de tráfego em uma via urbana na cidade de Calgary, no Canada. As vantagens, desvantagens e precisões desses métodos serão comparadas.

3.1.2 Dados de estudo

Os dados utilizados foram coletados em uma seção da rodovia e são disponibilizados na forma de Volume/hr e sentidos individuais. Essas informações foram extraídas no período de 1996 a 2000. A **Figura 3.1.a** representa o volume de tráfego no sentido sul-norte enquanto que a **Figura 3.1.b** representa o volume de tráfego no sentido norte-sul.

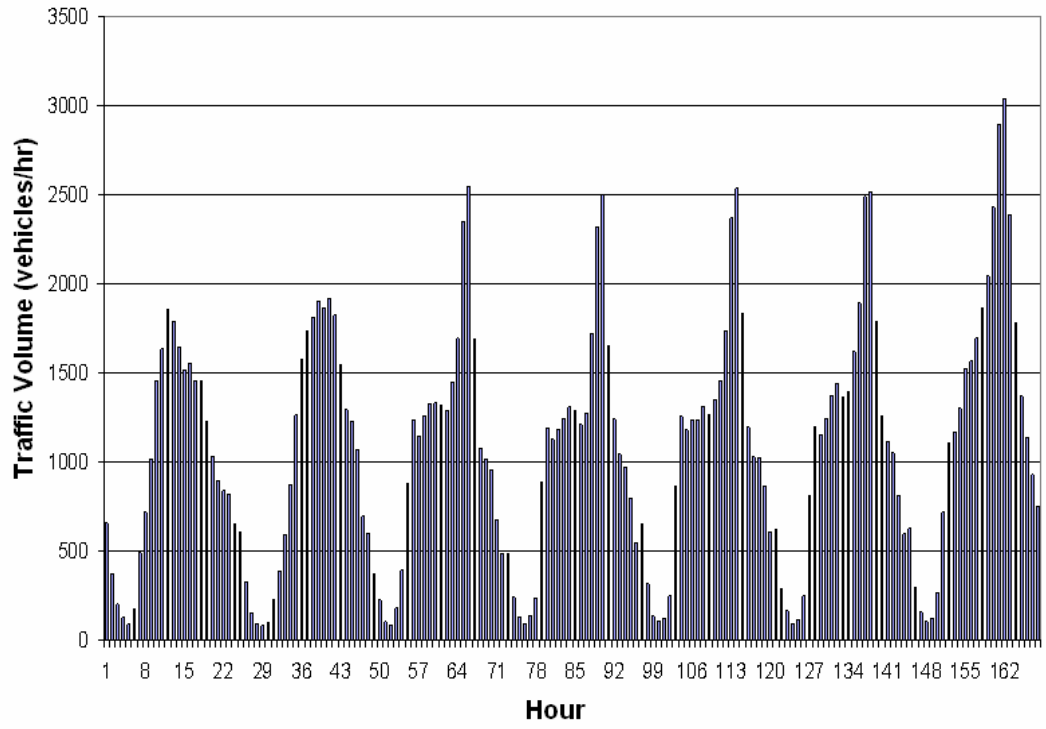

Figura 3.1.a Volume/h de tráfego de veículos no sentido sul-norte

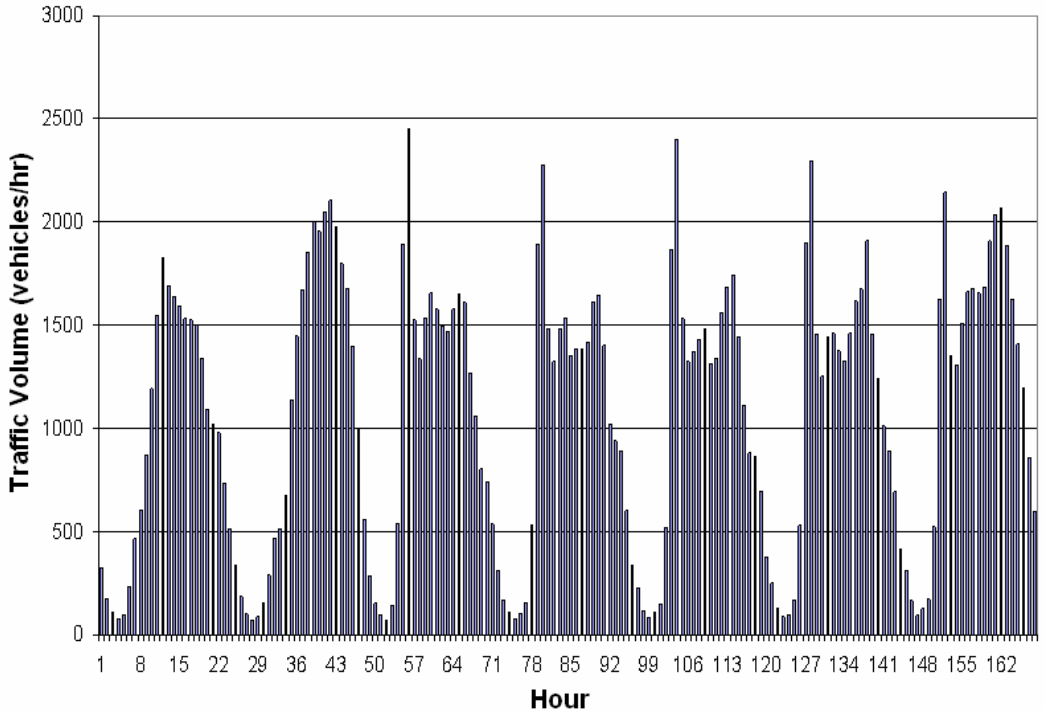

Figura 3.1.b Volume/h de tráfego de veículos no sentido norte-sul

No sentido norte-sul, destacam-se dois períodos de pico, sendo um pela manhã e outro à tarde. Enquanto que no sentido oposto apenas um período de pico se destaca no período da tarde. Essa diferença ocorre devido a passagem de veículos que percorrem longas distâncias utilizarem essa via, já que ela interliga pólos comerciais ao norte.

Como objetivo desse estudo é apresentada uma previsão de 12hrs para o volume de tráfego, sendo o período definido entre 8h00min e 20h00min.

3.1.3 Modelos analisados

Existem diversos métodos que podem ser utilizados na previsão de tráfego a curto prazo. Esses serão chamados de Modelos Naïve. Os modelos

Naïve trabalham de uma forma extremamente simples: eles geram valores para pesos a partir de dados históricos.

Stephanedes et al. (1981) propôs um modelo de média histórica para o controle de tráfego em tempo real. O algoritmo proposto era provido de uma capacidade superior aos algoritmos até então utilizados, como os algoritmos desenvolvidos em intervalos de controle na ordem de 5-15min (conhecidos como a segunda geração de algoritmos), e os embasados em mecanismos de ciclo-a-ciclo (conhecidos como a terceira geração) que apresentam como maior vantagem a simplicidade; por outro lado, eles oferecem uma precisão inferior a dos algoritmos mais sofisticados.

Koutsopoulos e Xu (1992) demonstraram que a simples reutilização de dados históricos como base para o controle de tráfego em tempo real é de precisão significantemente inferior ao uso de dados correntes e previsões futuras.

Os seguintes modelos Naïve são utilizados para estimar volumes em horas:

- Valor do Último Ano (VUA): reuso das ocorrências do último ano
- Valor do Último Mês (VUM): reuso das ocorrências do último mês
- Média Histórica: calculo da média baseando-se nos últimos 3 anos

3.1.4 Modelos de análise de séries temporais

- Uma série temporal é uma sequencia de observações de uma variável em particular. Elas são constantemente analisadas em busca de padrões históricos que sejam passíveis de reutilização em predições.
- A modelagem de séries temporais é embasada na premissa de que valores históricos de uma variável podem prover indicações de seus

possíveis valores futuros (Box e Jenkins 1970).

- Os modelos são a Média Móvel com Peso Exponencial (MMPE) e a Média Móvel Integrada Auto-regressiva (MMIA).

Na MMPE os valores históricos observados nas horas das últimas 12 semanas são utilizados para calcular uma estimativa segundo a equação abaixo:

$$X_i = (1 - ø)*X_{t-1} + (1 - ø)*ø*X_{t-2} + (1 - ø)*ø^2*X_{t-3} + ... + (1 - ø)*ø^{n-1}*X_{t-n}$$

Onde $\mathbf{X_{t-1}}$ representa a observação mais recente; $\mathbf{X_{t-n}}$ representa a observação mais antiga; ø define o grau de filtragem, sendo uma constante definida a partir dos dados e limitada entre $0 < ø < 1$.

Já na MMIA os padrões de volume de tráfego das últimas oito ocorrências de um dia da semana (ex: as últimas oito quartas-feiras) são utilizados para desenvolver o modelo e estimar volumes durante as 12 horas da nonagésima ocorrência (ex: a nonagésima quarta-feira).

3.1.5 Modelos de Redes Neurais por defasagem de tempo geneticamente desenvolvidos

Uma das variantes de Redes Neurais conhecida como Rede Neural por Defasagem de Tempo (RNDT) é estudada com maiores detalhes neste trabalho.

Essa é composta por três camadas, são elas a camada de entrada, oculta e de saída. Cada camada tem um ou mais neurônios. A RNDT é de grande interesse para análise de séries temporais. O principal motivo é que os neurônios que compartilham uma mesma camada podem receber informações com defasagem de tempo de outros neurônios também pertencentes a ela. Supondo que um neurônio de uma dada camada receba uma informação do meio externo e a propague para o neurônio a sua direita após um intervalo previamente definido, como ilustra a **Figura 3.2,** a camada de entrada irá manter

uma parte da série temporal. Esse mesmo mecanismo também pode ser incorporado em outras camadas.

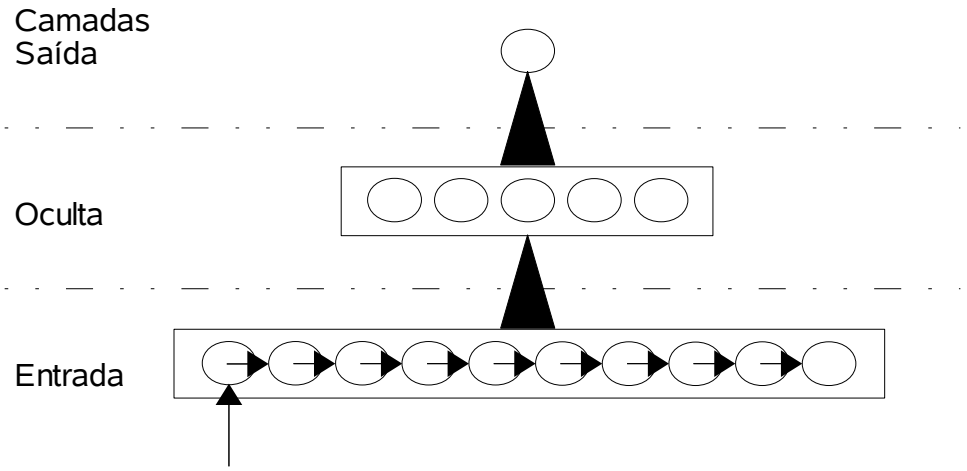

Figura 3.2 Estrutura da Rede Neural por defasagem de tempo.

Esse estudo foi estruturado na proposta apresentada por Lingras et al. (2002) devido a sua simplicidade e praticidade de execução. Inicialmente o algoritmo desenvolvido sob uma arquitetura genética tem acesso ao conjunto de dados da primeira semana, composto por 168 volumes, observados em cada uma das vinte e quatro horas de cada um dos sete dias (7 dias * 24 horas = 168). Somente então o algoritmo selecionará 24 entradas definitivas. Presume-se que esse conjunto de 168 volumes contenha as informações necessárias para a predição da próxima hora. O motivo pela escolha de 24 das 168 entradas é que experimentalmente a adição de mais entradas implicam pouca ou nenhuma melhora na precisão do algoritmo.

O critério utilizado na seleção das entradas é de que elas tenham a máxima correlação com o volume da hora preditada, dentre todas as combinações possíveis ($C_{168,24}$). Então, o conjunto final de entradas é inserido na RNDT para a predição.

Alguns parâmetros foram pré-definidos, dentre eles o tamanho populacional do Algoritmo Genético (AG) em 110. Os indivíduos podem se

desenvolver por até 1000 gerações. Para mutação, um operador de substituição randômico é utilizado e a probabilidade de mutação é fixada em 1%. Um operador de cruzamento simples é utilizado, sendo este definido a 90%. Esses valores foram selecionados após múltiplos experimentos com diferentes valores.

O cromossomo usado como solução é o último cromossomo criado após um ciclo de 1000 gerações. As conexões selecionadas pelo Algoritmo Genético foram utilizadas para o desenvolvimento e implementação do modelo neural.

Existem 168 neurônios na camada de entrada do modelo RNDT, porém somente 24 devem ter conexão com a camada oculta, que é formada por 12 neurônios. A camada de saída possui apenas um neurônio.

Todos os modelos foram testados e treinados. O percentual de erro absoluto foi calculado da seguinte forma:

$$P_{ea} = \frac{volAtual - volEstimado}{volEstimado}$$

Os parâmetros chave da comparação consistem na média do erro e o erro do 95° percentil. Essas medidas dão uma boa ideia da distribuição de erros ao incluir (ao calcular-se a média dos erros) ou ao excluir (ao calcular-se o 95° percentil do erro) muitos dos erros causados por eventos especiais.

3.1.6 Resultados

Os modelos anteriormente citados passaram por experimentos práticos para a previsão de 12hrs de tráfego em dois dos dias de uma semana, às quartas-feiras e aos domingos nos meses de julho e agosto.

Os resultados gerados para os erros das quarta-feiras e dos domingos

foram essencialmente os mesmos; por esse motivo apenas serão apresentados os resultados das quarta-feiras.

Para facilitar a comparação, os valores de erro médio e o erro de 95° percentil para cada direção serão apresentados nas mesmas tabelas.

Os maiores erros foram resultados do Valor do Último Ano no sentido sul-norte, enquanto que os menores erros foram resultados no sentido oposto. O motivo para essa discrepância é um aumento drástico no volume de tráfego entre o ano anterior (de onde os dados se originaram) para a situação atual. O maior erro médio no sentido norte foi 34,1% e o 95° percentil foi de 81,7% durante as horas de pico do período diurno (entre 7h00min e 8h00min). Também resultou em grande erro o período de pico da tarde (entre 16h00min e 18h00min): o erro médio foi superior a 20%. Contrastando com esses resultados, o tráfego no sentido oposto foi relativamente estável, resultando em um erro médio entre 5-7% e um 95° percentil entre 11-15%.

Já o modelo de Média Histórica apresentou um resultado mais preciso, com maior proximidade entre os resultados dos diferentes sentidos. O erro médio varia entre 9 e 12% e o 95° percentil entre 13 e 18%.

O modelo de Valor do Último Mês teve bons resultados para o erro médio, mantendo-o entre 5 e 8%, exceto por alguns períodos no sentido norte-sul. Por outro lado, o erro do 95° percentil ficou entre 17 e 18%.

A **Tabela 3.1** mostra os erros na previsão dos modelos Média Móvel com Peso Exponencial e Média Móvel Integrada Auto-regressiva respectivamente. Se comparados com os modelos Naïve, esses tem uma precisão superior.

Horário	Erros em Predição							
	Erro Médio				95° Percentil			
	MMPE		MMIA		MMPE		MMIA	
	n	**s**	**n**	**s**	**n**	**s**	**n**	**s**
07-08	05.00	09.50	05.40	05.30	09.80	12.90	09.90	09.30
08-09	03.90	07.00	04.60	03.10	10.10	12.30	08.70	05.10
09-10	05.40	05.50	07.50	04.30	14.00	13.50	13.20	06.60
10-11	05.00	04.30	04.50	01.70	15.80	14.90	07.40	02.90
11-12	05.90	04.80	04.00	05.10	16.90	16.10	05.90	09.30
12-13	04.50	05.10	02.70	05.00	13.10	17.10	04.70	09.60
13-14	04.90	05.70	05.00	06.20	10.80	15.80	10.50	12.20
14-15	02.90	04.30	05.10	03.00	08.90	15.30	10.50	06.20
15-16	03.40	03.10	03.80	02.90	08.90	08.80	07.00	04.00
16-17	04.50	03.10	02.50	04.00	09.50	08.60	04.40	06.40
17-18	05.50	03.40	03.50	07.80	11.00	12.60	06.80	15.00
18-19	05.40	05.00	06.90	04.00	15.10	13.70	13.30	06.90
Média Total	04.70	05.10	04.60	04.40	12.00	13.50	08.50	07.80

Tabela 3.1 Comparação de erros entre MMPE e MMIA

As médias dos erros médios para esses modelos estão entre 4 e 5.1%. As médias para os erros dos 95° percentis estão entre 12 e 13.5% para o modelo MMPE e entre 7.8 e 8.5% para o modelo MMIA. Os valores das médias totais mostram que o modelo MMIA tem, na maioria das vezes, resultados mais precisos que o modelo MMPE.

Horário	Erros em Predição			
	Erro Médio		95° Percentil	
	n	**s**	**n**	**s**
07-08	04.40	03.07	10.40	06.68
08-09	02.92	02.63	07.31	05.11
09-10	05.70	01.96	08.12	05.29
10-11	04.10	04.75	09.90	11.12
11-12	04.01	04.42	07.75	09.56
12-13	03.79	02.94	08.38	06.22
13-14	04.56	03.75	08.10	08.63
14-15	06.21	03.90	10.22	07.88
15-16	06.35	04.97	10.86	07.58
16-17	04.80	06.84	06.96	11.81
17-18	07.61	05.89	11.92	10.38
18-19	05.80	09.02	11.91	21.09
Média Total	05.02	04.51	09.32	09.28

Tabela 3.2 Erros no modelo RNDT

Utilizando as entradas definitivas, geradas pelo algorítimo geneticamente estruturado, na RNDT executou-se uma etapa de treinamento a partir de um conjunto de dados. Em seguida, os modelos treinados foram utilizados para fazer as previsões.

A **Tabela 3.2** mostra os erros durante a fase de teste dos modelos RNDT. O erro médio nos modelos RNDT estão, em sua maioria, entre 4 e 5% e no 95° percentil entre 9 e 10%.

A análise mostra que o modelo RNDT tem precisão comparável ou superior aos modelos MMIA. Os erros médios quando comparados entre os modelos, o RNDT e o MMIA mantem um equilíbrio no sentido norte. Já no sentido oposto, sul, o modelo RNDT supera o MMIA em aproximadamente 58% dos casos. Outro destaque importante é que o modelo RNDT é capaz de se

adaptar a constantes mudanças do ambiente, sendo esse um diferencial quanto a precisão em campo.

3.2 Sistema adaptativo baseado na computação evolucionária e Redes Neurais para a previsão de tráfego a curto prazo (Annunziato et al. 2004)

A proposta deste estudo é apresentar uma solução que combine algoritmos de computação evolucionaria com Redes Neurais de propagação reversa na previsão do tráfego urbano (Annunziato et al. 2004). Suplementarmente será apresentada uma comparação dos resultados entre o clássico algoritmo de propagação reversa e de sua combinação com algoritmos evolucionários.

3.2.1 Análise de dados e configuração neural utilizada

A entrada do modelo neural foi escolhida por meio da metodologia de análise dinâmica não-linear nos sinais dos fluxos de tráfego. Maiores detalhes sobre essa metodologia podem ser encontrados em Abarbanel (1996) e Annunziato e Abarbanel (1999).

A pesquisa é baseada na utilização de n amostras do sinal x(t), x(t-T), ..., x{t-T(n-1)} onde T é o tempo característico necessário para preservar a informação requerida referente às dinâmicas. A escolha pelos melhores n e T é dependente do sinal. O sinal representa a taxa de fluxo de tráfego da última hora com as amostras extraídas a cada 6 min. Análises mostram que os melhores valores para T e n são respectivamente entre 10 e 15 min e 6 e 10 min; então T

foi definido como sendo 12 min, o que corresponde a 2 amostras e n sendo 8.

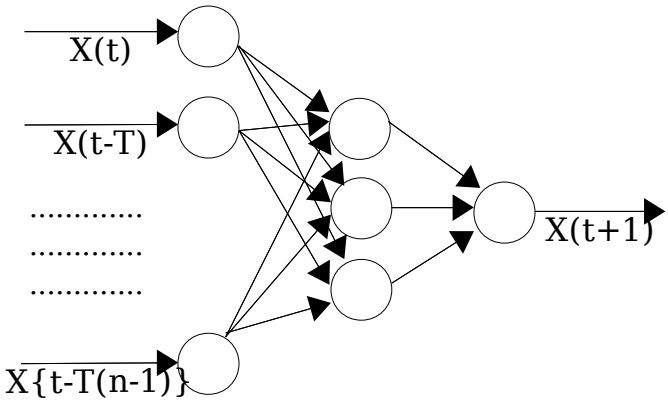

Figura 3.3 Topologia Neural.

Dessa forma, a Rede Neural será composta por 8 neurônios de entrada, 3 na camada oculta e 1 na camada de saída como mostra a **Figura 3.3**.

3.2.3 Ambiente do sistema adaptativo

Esse estudo visa mostrar como algoritmos evolucionários podem ser empregados na otimização dos pesos de algoritmos de propagação reversa *on-line* bem como reportar os resultados da previsão de tráfego a curto prazo.

A otimização *on-line* funciona como ilustrado na **Figura 3.4.** O conceito básico é construir um ambiente artificial que exerça uma atividade em paralelo com o processo real.

Supõe-se que ocorra a execução constante da leitura de dados do processo real e posterior atualização de um conjunto de dados, representando este o objeto da otimização. Uma vez nessa situação, *performance* poderá ser definida como a capacidade de cada Rede Neural de reconstruir um conjunto de dados reais porém não imediatamente óbvios.

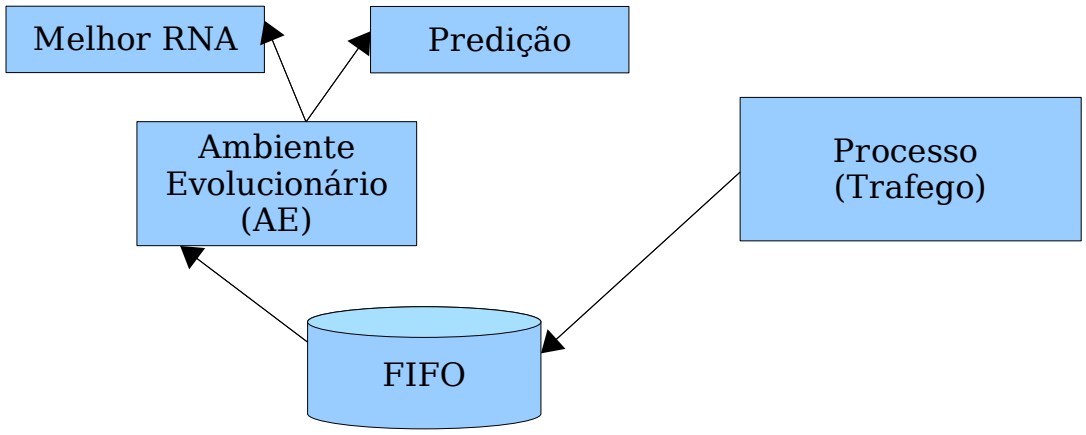

Figura 3.4 Sistema adaptativo.

A estratégia utilizada na atualização dos dados é primeiro a entrar primeiro a sair (também conhecido como *first in first out),* objetivando a previsão em uma janela de tempo onde o seu tamanho depende do problema a ser tratado.

Nessa situação, sempre que um conjunto de dados muda, uma nova rede é dinamicamente definida; para isso, tem-se um modelo evolucionário capaz de se adaptar e seguir em tempo real a evolução do Processo. Dessa forma o ambiente evolucionário continuamente provê a melhor Rede Neural correspondente a um teste de resistência de um indivíduo e a previsão da taxa de fluxo de tráfego relativa.

Nesse contexto cada indivíduo do ambiente evolucionário representa uma Rede Neural de propagação reversa, competindo com as outras por meio de testes de resistência, tendo como genótipo os pesos sinápticos.

A resistência é medida através do erro global na modelagem do banco de dados de treinamento a partir da seguinte fórmula:

Resis = 1 – EQM

Onde EQM significa o Erro Quadrático Médio normalizado entre 0 e 1 e utilizado pelo algoritmo de propagação reversa. Esta função é utilizada para fazer uma comparação direta entre os resultados obtidos com os gerados pela metodologia de propagação reversa.

EQM = $\sqrt{(1/n)*\Sigma(0.5*\Sigma(Y-Yt)^2)}$

Onde **n** representa a dimensão do conjunto de dados de treinamento, m o número de neurônios de saída (nesse caso m=1), Y o valor de saída estimado e Yt o valor de saída esperado. Todas as entradas e saídas da rede são normalizadas entre 0 e 1. Para medir o nível encontrado é lida a resistência do melhor indivíduo (correspondente a Rede Neural de maior performance). O algorítimo utilizado para desenvolver a Rede Neural Evolucionária foi o População Caótica (2002) *(Chaotic Populations)*(Annunziato e Pizzuti 2002*)*.

3.2.4 Resultados experimentais

Os experimentos foram focados em prever a taxa de fluxo de tráfego de 24 e 60 minutos utilizando RNAs. Utilizou-se o algoritmo de propagação reversa no treinamento *off-line* e o algoritmo combinado no treinamento *on-line*. O objetivo é otimizar dinamicamente os pesos de conexões de uma Rede Neural. Em ambas as situações foi utilizada a função senoidal:

Y = $1/(1+e^{-x})$

Os dados de treinamento consistem de uma semana de observação, 1650 amostras reais extraídas a cada 6 minutos representando a taxa de fluxo de tráfego da última hora em quatro pontos diferentes localizados na cidade de

Terni, distando 100km de Roma, com aproximadamente 100000 habitantes e uma rede composta por 87 sensores.

A primeira sequencia de resultados refere-se ao algoritmo de propagação reversa trabalhando de forma *off-line*. Para tal, experimentos um milhão de ciclos foram configurados.

O segundo grupo de testes envolve a otimização *on-line*. Nesse estágio foi considerada uma janela de tempo de 1 hora, representando 50 gerações evolucionárias, correspondendo ao tempo necessário para se adaptar às novas situações, e uma população máxima de 50 unidades. O algorítimo de População Caótica (Annunziato e Pizzuti 2002) não precisa de configurações adicionais. Para comparar os dois modelos, os dados foram particionados em dois conjuntos, sendo um de 1500 pontos para o treinamento e outro de 150 pontos para testes.

Para o modelo *off-line,* o conjunto de treinamento foi utilizado por completo e posteriormente, após a fase de aprendizagem, o conjunto completo de testes. No modelo *on-line,* as Redes Neurais são interativamente criadas como o resultado da adaptação do algoritmo de População Caótica (Annunziato e Pizzuti 2002) no decorrer da janela de tempo, tornando-se assim possível a predição do próximo ponto.

A **Tabela 3.3** a seguir apresenta uma média, de acordo com a equação do Erro Quadrático Médio dos quatro sensores considerados.

	Predição 24 min		Predição 60 min	
	Treinamento	Teste	Treinamento	Teste
Propagação Reversa	0.040	0.063	0.061	0.169
Rede Neural Evolucionaria	0.033	0.054	0.036	0.068

Tabela 3.3 Comparação dos erros quadráticos médios

A **Tabela 3.4** apresenta os resultados de cada um dos sensores individualmente nos tempos de predição de 24 e 60 minutos.

	Predição 24 min		Predição 60 min	
	Treinamento	Teste	Treinamento	Teste
Sensor 1				
Propagação Reversa	0.040	0.060	0.059	0.118
Rede Neural Evolucionaria	0.030	0.040	0.035	0.064
Sensor 2				
Propagação Reversa	0.036	0.065	0.059	0.216
Rede Neural Evolucionaria	0.030	0.058	0.035	0.066
Sensor 3				
Propagação Reversa	0.046	0.06	0.071	0.180
Rede Neural Evolucionaria	0.038	0.06	0.039	0.078
Sensor 4				
Propagação Reversa	0.037	0.065	0.054	0.163
Rede Neural Evolucionaria	0.032	0.059	0.035	0.064

Tabela 3.4 Comparação dos erros quadráticos médios por individuo

As tabelas **3.3** e **3.4** trazem a comparação entre os algoritmos de Propagação Reversa e a combinação desses com algoritmos Evolucionários formando as Redes Neurais Evolucionárias. Os resultados obtidos mostram a redução nos erros quadráticos médios no algoritmo de Rede Neural Evolucionaria. A única exceção ocorre na analise individual dos sensores especificamente no teste de predição de 24 min no Sensor 3.

3.3 Rede Neural com alimentação reversa e evolucionária para a previsão de tráfego (Annunziato et al. 2003)

3.3.1 Descrição

O escopo desse estudo é otimizar o algoritmo de propagação reversa por meio de algoritmos evolucionários e posteriormente comparar os resultados das combinações com os resultados do algoritmo de propagação reversa sem a otimização (Annunziato et al. 2003). Esses algoritmos devem prever o fluxo de tráfego com 20 minutos de antecedência.

3.3.2 Processamento dos dados

O sinal do fluxo de tráfego é dinâmico e não linear e quando exportado para uma forma gráfica esse pode ter uma forma parecida com a **Figura 3.5.** Essa figura mostra um gráfico, a título ilustrativo, reconstruído em duas dimensões que reporta os sinais de fluxo de tráfego em múltiplos dias de uma semana. Esse gráfico foi gerado usando-se duas amostras de tempos espaçados do sinal de fluxo de tráfego: x(t) e x(t+T) onde T representa a característica defasagem de tempo que permite o desenvolvimento do gráfico. Um período entre 5 e 15 minutos como o tempo de defasagem na reconstrução do gráfico é considerado o suficiente para o seu desenvolvimento. Isso quer dizer que dados coletados nesse período devem ser capazes de disponibilizar uma descrição mínima da dinâmica do sinal. Com base neste fundamento, o tempo de defasagem foi fixado em 10 minutos.

O sinal do fluxo de tráfego é medido uma vez a cada minuto e essa informação é então disponibilizada para o tratamento. Levando em conta o

processamento de 10 amostras, é possível trabalhar com sub-amostras para que haja uma redução nos tempos computacionais.

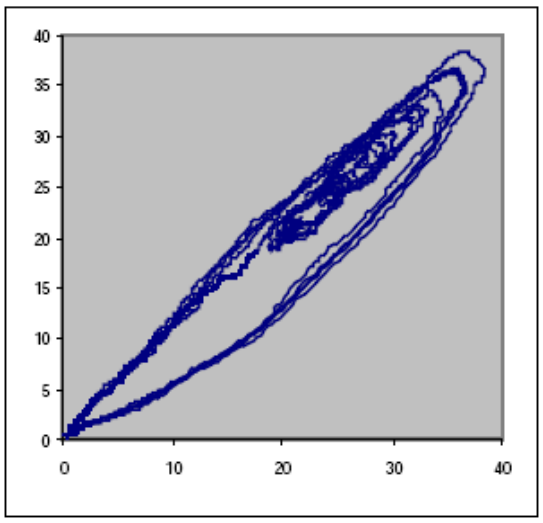

Figura 3.5 Gráfico do sinal de fluxo de tráfego de segunda a sexta-feira.

Por essa razão é aplicado um filtro de média móvel linear de 5 minutos e então uma nova amostra é retirada do sinal num período de 5 minutos. Finalmente são usadas como entradas as séries de amostras retiradas a cada 10 minutos de acordo com o tempo de defasagem. A rede será formada por oito neurônios de entrada, uma única camada oculta contendo três neurônios e um neurônio na camada de saída.

3.3.2 Os algoritmos evolucionários

Dois algoritmos evolucionários foram analisados e utilizados em conjunto com os algoritmos de propagação reversa visando a obtenção de resultados mais precisos na previsão a curto prazo.

O principio básico de ambos os algoritmos é ter um certo grau de

liberdade no desenvolver de seu comportamento ao combinar a genética com outros aspectos peculiares da vida como interação, competição, cooperação, necessidade de alimentação etc. Nessa proposta, cada indivíduo representa uma rede de propagação reversa, em constante competição com outras por meio do teste de resistência que representa, por sua vez, a capacidade que cada indivíduo possui de reconstruir seu banco de treinamento tendo como genótipo os pesos sinápticos.

Ambos os algoritmos tem como parâmetros pré-definidos o número de gerações e o tamanho máximo populacional. Todos os outros parâmetros relacionados, como operador de cruzamento simples e mutação, dependem de dinâmicas internas.

O teste de resistência é medido por referência ao erro global na modelagem do banco de treinamento de acordo com as seguintes fórmulas anteriormente citada (item 3.2.3):

Resis = 1 – EQM

EQM = $\sqrt{(1/n)*\Sigma(0.5*\Sigma(Y-Yt)^2)}$

3.3.3 Resultados experimentais

O foco do estudo é prever os níveis de tráfego com uma antecedência de 20 minutos, utilizando-se para isso uma rede composta por oito neurônios de entrada, três na camada oculta e um neurônio de saída. Os algoritmos utilizados serão o de propagação reversa, a combinação entre o algoritmo de propagação reversa e o evolucionário da População Caótica e a combinação entre propagação reversa e o evolucionário Evolução Parcial. A função de transferência será a:

Y = $1/(1+e^{-x})$

Os dados de entrada utilizados consistem de uma semana de observação. Esses foram divididos em 1800 dados para treinamento e 180 dados para os testes.

	Propagação Reversa	População Caótica	Evolução Parcial
off-line	0,100	0,043	0,038
on-line		0,013	0,005

Tabela 3.5 Comparação de erros de treinamento

	Propagação Reversa	População Caótica	Evolução Parcial
off-line	0,110	0,042	0,035
on-line		0,031	0,033

Tabela 3.6 Comparação de erros de testes

	Gerações/Ciclos	Máxima densidade populacional
Propagação Reversa	3000000	
População Caótica	500	100
Evolução Parcial	2000	256

Tabela 3.7 Configuração utilizada no treinamento *on-line*

	Gerações	Máxima densidade populacional
População Caótica	500	100
Evolução Parcial	2000	256

Tabela 3.8 Configuração utilizada no treinamento off-*line*

As tabelas **3.5** e **3.6** mostram os resultados obtidos pelos algoritmos: Propagação Reversa, combinação Propagação Reversa e População Caótica e combinação Propagação Reversa e Evolução Parcial respectivamente. Dentre os resultados a menor incidência de erros ocorreu na execução da combinação de algoritmos de Propagação Reversa e Evolução Parcial. Já as tabelas **3.7** e **3.8** descrevem as configurações utilizadas durante a execução dos algoritmos.

3.4 Um algoritmo de propagação reversa aplicado a medição de congestionamento (Fotouhi et al. 2005)

3.4.1 Descrição

Esse estudo (Fotouhi et al. 2005) apresenta uma solução de processamento distribuído para algoritmos de propagação reversa fazendo-se uso de um *cluster* de PCs, podendo cada um deles ser acessado por meio de uma aplicação baseada em MPI (*Message Passing Interface*).

A eficiência dessa implementação foi testada na classificação de padrões do problema de congestionamento de tráfego urbano. Os resultados foram comparados com os algoritmos utilizados de forma serial.

Em Petrowski et al. (1993), os autores especificam três esquemas para uma implementação da metodologia de propagação reversa de forma paralela:

- Mapeamento de cada nó/máquina a um processador: dessa forma a máquina de processamento paralelo se torna um modelo físico de rede.
- Divisão da matriz de pesos entre os processadores permitindo que um segmento do vetor de entrada possa ser operado a qualquer momento.
- Copiar toda a rede em cada processador permitindo o treinamento sequencial da rede para uma porção do conjunto de treinamento.

Considerando a implementação de uma grande Rede Neural com base em um ambiente MPI, o primeiro esquema torna-se impraticável dado ao fato que a quantidade de nós seria significantemente maior que a quantidade de processadores.

O segundo esquema é interessante para uma arquitetura de compartilhamento de memória SIMD. Se o algorítimo de propagação reversa for implementado por MPI, teria que haver um mecanismo de agendamento dedicado a distribuição dos subconjuntos de pesos entre as camadas entrada-oculta e oculta-saída.

Já a terceira poderia ser facilmente simulada por um sistema distribuído MPI com uma pequena mudança no algoritmo sequencial.

O mecanismo de simulação paralela usando MPI consiste em dividir e distribuir subconjuntos de padrões entre os processadores, aprender um conjunto de padrões específicos e recolher os conjuntos de matrizes de pesos.

A **Figura 3.6** mostra o mecanismo de uma implementação do algoritmo de propagação reversa sobre MPI em uma arquitetura paralela e distribuída. Dependendo do conjunto de dados disponibilizado, o período de final de aprendizado de cada processador pode ser diferente.

A condição final pode ser uma entre as seguintes:

- O erro final está abaixo do erro esperado;
- O número de interações está acima de um limite máximo determinado.

É importante distribuir de forma equalizada um subconjunto de padrões para cada processador. Isso implica que um dado subconjunto de padrões pode conter toda funcionalidade de um dado conjunto de padrões.

Se um processador aprende a rede com um conjunto de padrões desbalanceado e distribuído, o treinamento pode não ter utilidade. Nesses casos, será necessário o reaprendizado de todo o conjunto de padrões. O

aprendizado não certifica a melhor solução. Depois de terminada a fase de aprendizado em cada processador, um processador primário recupera as matrizes de pesos de todos os processadores. E assim faz a média das matrizes cuja somatória dos erros quadráticos seja a mínima e a utiliza na fase do reaprendizado.

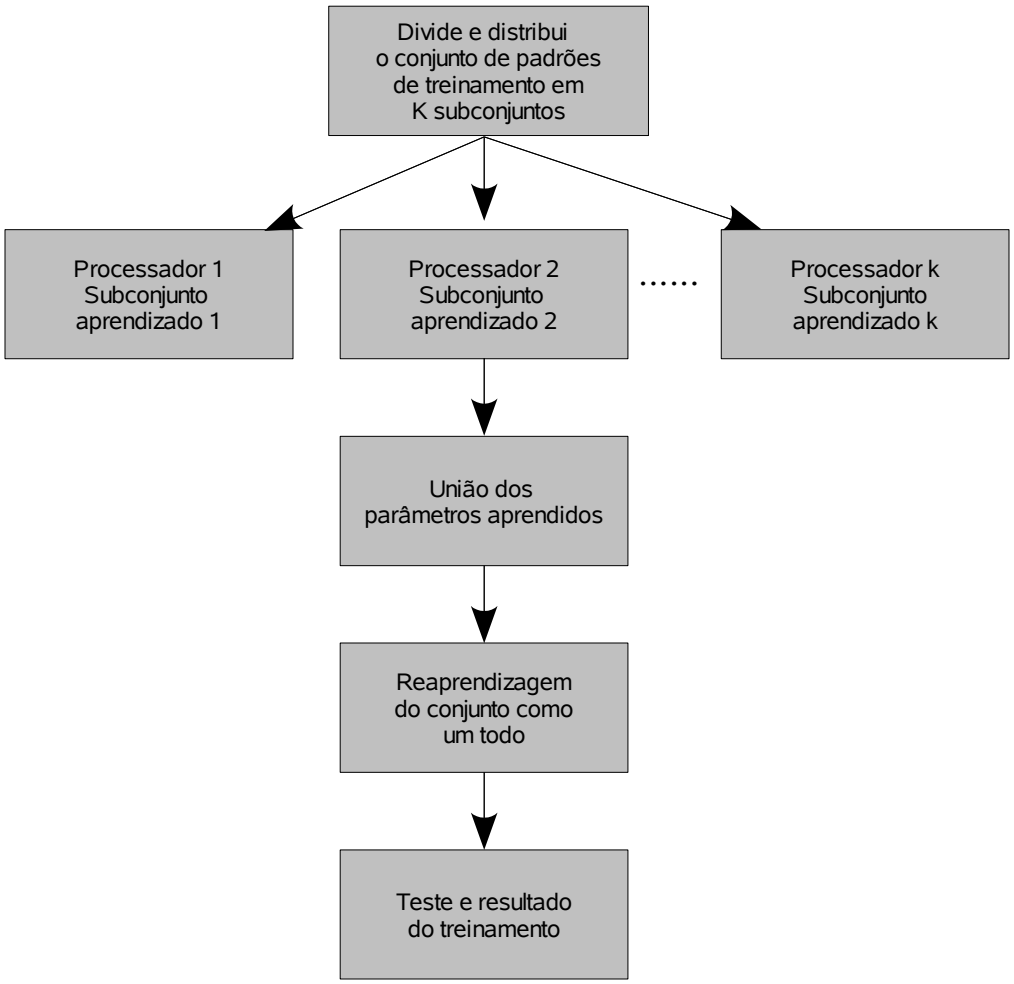

Figura 3.6 Mecanismo de aprendizagem paralela.

3.4.2 Dados de tráfego

Foi empregado como estratégia para a detecção do tráfego, um Radar, também conhecido como Sensor Micro-ondas de tráfego Remoto. Esse radar é capaz de detectar o volume de veículos, a velocidade e ocupação por faixa.

O radar pode estar disposto em uma das duas formas, como monitoramento de multi-faixas ou como detector de velocidade e dimensão de congestionamento em uma única faixa.

Os dados foram coletados de duas formas: por meio do radar, a cada 30 segundos, e também manualmente, a cada 5 minutos em um período de 18 dias. Os dados coletados foram a velocidade média, o volume e a média de ocupação por faixa.

Congestionamentos são geralmente classificados pela velocidade dos veículos como um todo. Nesse caso a classificação foi dividida em 3 níveis:

- Congestionamento de 0 a 10km/h;
- Lentidão entre 10 e 30km/h;
- tráfego fluente acima de 30km/h.

A categorização de congestionamento é complexa devido a dificuldade encontrada ao medir a velocidade do tráfego como um todo. Um método de implementação de detectores de tráfego foi apresentado por Yoshizaki (1995). Nesse método, detectores de tráfego foram instalados próximos uns aos outros e com intervalos regulares entre 100 e 300 metros respectivamente.

A **Figura 3.7** mostra o posicionamento dos detectores de tráfego e a relação entre períodos de tráfego e tamanho das filas (congestionamento).

Baseado no tamanho da fila, o tráfego pode ser classificado em três diferentes níveis. Assume-se, como premissa, que a maior parte dos veículos consigam trafegar por esse trajeto em três períodos do sinal.

O primeiro nível ocorre se os veículos localizados na marca dos 200m podem cruzar a área de intersecção durante um período do sinal.

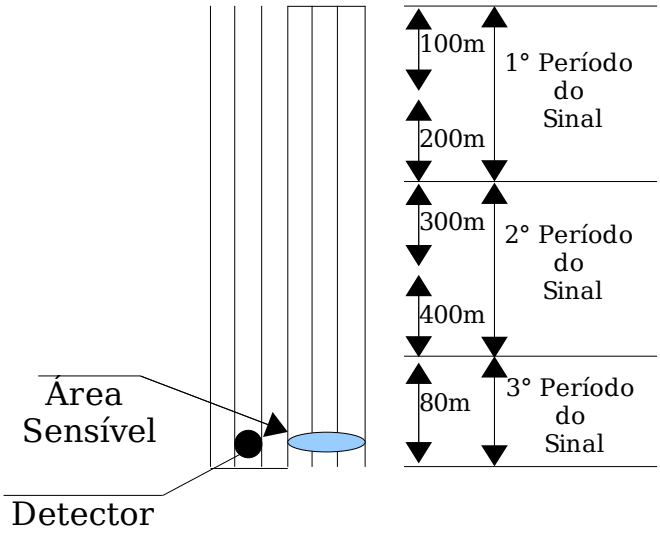

Figura 3.7 Relação entre os períodos de tráfego.

Se um período do sinal dura 120 segundos e o nível de congestionamento for o 2°, então o tempo de passagem nesse trecho é de 240 segundos.

3.4.3 Experimentos e resultados

Experimentos foram conduzidos para testar a Rede Neural paralela. Foram simulados dois casos de classificação, o de 2 e o de 3 níveis.

Os padrões de entrada foram adicionados a cada 5min para corresponderem com os padrões de saída. Cada dado de entrada foi normalizado em 0 ou 1. As dimensões das camadas são: camada de entrada 9 neurônios, camada oculta 18 e camada de saída 2 e 3 neurônios de acordo com

o modelo a ser analisado. O conjunto de treinamento do modelo de duas classes foi preparado de forma a agrupar os dados de lentidão e tráfego fluente em uma única classe.

Dados de tráfego coletados nos 13 primeiros dias foram utilizados para o treinamento enquanto que os dados coletados nos últimos 5 dias foram usados para o teste.

Os dados coletados em dias consecutivos foram organizados em 24 conjuntos. O conjunto de treinamento consiste em 312 padrões, enquanto que o de teste consiste em 120 padrões. Para o experimento, as saídas iguais ou superiores a 0,6 foram normalizadas para 1 e as inferiores para 0. A função de ativação utilizada foi a senoidal.

Número de classes	Treinamento (312)	Teste (120)
2	94.8%	93.3%
3	92.9%	91.7%

Tabela 3.9 Comparação das porcentagens de acertos

O algoritmo de aprendizagem responsável pelas classificações ilustradas na **Tabela 3.9** foi o de propagação reversa. A simulação foi feita com dois processadores. No caso de usar mais de 3 processadores os dados seriam insuficientes para ocorrer a convergência, se comparados com a complexidade do problema.

Uma discrepância faz-se perceptível quando comparados os resultados dos modelos de duas e três classes.

3.5 Uso de sistema multi agente para avaliar estratégias de decisão de controle de tráfego urbano

3.5.1 Descrição

O desenvolvimento econômico e os meios de transporte estão fortemente ligados. Já em 1969 foi relatado por Stern que o ritmo da industria automobilística nacional acarretará uma saturação das vias de centros urbanos mal projetados e de capacidade limitada com relação ao número de veículos que entram em circulação.

Para auxiliar o controle de trânsito são utilizados os sinais de trânsito (semáforo). Dada a considerável oscilação no volume de veículos que trafegam nos centros urbanos, os sinais de trânsito tornam-se insuficientes para gerenciar a demanda. Visando a otimização dos sistemas seria necessário um sincronismo na rede de semáforos; porém, o administrador do tráfego de uma cidade se depara constantemente com a seguinte questão:

O sistema proposto nesse estudo (Hübner e Schmitz 2002) tem como objetivo auxiliar na decisão do administrador de trânsito especialmente quando deparado com esta questão: dentre as várias estratégias de sincronização de sinais de trânsito, qual seria a de maior empregabilidade e que gerencie as peculiares situações da cidade?

Considerando que as principais características do problema são a distribuição e a constante alteração das informações necessárias à administração do controle de trânsito, o desenvolvimento de uma ferramenta centralizada é bastante difícil. Por tanto, tem-se como hipótese que esse problema possa ser simplificado por meio de uma modelagem segundo o paradigma de Sistemas Multi Agente (SMA). Em um SMA, um problema é resolvido dividindo-se o trabalho entre múltiplos agentes autônomos que

trabalham em um ambiente cooperativo onde interagem e trocam conhecimentos sobre um problema em foco e sua solução (Weiss 1999 e Bordini 2001).

3.5.2 Tráfego urbano

Existem algumas características que devem ser observadas para a gerência de trânsito. Como a classificação do fluxo de tráfego (interrompido e interrupto), tipos de cruzamentos (cruzamentos em nível e cruzamento em duplo) e tipo de sinalização (advertência e regulamentação). Mais informações sobre essas classificações podem ser obtidas em Ejzenberg (1996), Espel (2000) e Lopes (1998).

Esse estudo enfatiza os tempos semafóricos. O Código de Trânsito Brasileiro determina que para a sinalização de veículos existem 3 cores, que são estabelecidas com suas respectivas indicações: vermelho, indica obrigatoriedade de parar; amarelo, indica atenção, devendo o condutor parar o veículo, salvo se isto resultar em situação de risco; verde, indica permissão de passagem, podendo o condutor efetuar operações indicadas pelo sinal luminoso, desde que respeitadas as normas gerais de circulação e conduta.

Segundo Espel (2000), a programação dos tempos semafóricos consiste na obtenção dos valores dos tempos de ciclo para a elaboração dos planos de tráfego necessários para a operação dos cruzamentos; sendo assim, cada fase de um semáforo tem um tempo equacionado por fórmulas, conforme observado em Vilanova (1998).

A utilização do sinal luminoso amarelo é dada pelo fato de não ser possível uma frenagem ou rompimento da inércia de um veículo em movimento de forma espontânea imediata, além de haver um tempo para a percepção humana; em Vilanova (1985), para um veículo que se desloca em velocidade constante de 40km/h, ou 11m/s, o motorista leva em média 1 segundo para

perceber e reagir ao sinal. Para a desaceleração do veículo é recomendado um valor de 2.8m/s. O tempo do amarelo deve ser obtido de acordo com a seguinte equação:

$T_a = T_p + V/D*2$

Onde T_a representa o tempo do amarelo, T_p o tempo da percepção do motorista, V a velocidade e D a desaceleração.

Nesse caso obtém-se um tempo de amarelo de 2.96 segundos e por medida de segurança o arredondamento é feito para 3 segundos.

Para a luz vermelha é necessário o cálculo de um tempo, tal que, permita que um veículo que tenha iniciado um cruzamento possa completá-lo com segurança. Sendo o comprimento de um veículo 40m, sua velocidade de 40km/h e um cruzamento de 15m esse tempo é igual a 1,72 segundos; com arredondamento para 2 segundos por segurança, esse valor pode ser adquirido a partir da seguinte fórmula:

$T_v = (C_c + C_v) / V$

Onde T_v é o tempo do vermelho, C_c o comprimento do cruzamento, C_v o comprimento do veículo e V a velocidade do veículo.

A função do sinal luminoso de cor verde é escoar o volume de veículos acumulados durante a luz vermelha. Segundo Stern (1969), o equacionamento deve levar em conta os seguintes valores pré-definidos: tempo de atraso do primeiro veículo, T_a = 3 segundos; intervalo de tempo entre o arranque de dois veículos sucessivos $Ä_t$ = 1,5 segundos; a distância percorrida durante o período de embalagem C_u = 60 metros; velocidade média durante o período de embalagem V_m = velocidade da via dividido por 2; comprimento médio de um veículo C_v = 4 metros; Com esses valores o tempo de escoamento é obtido por:

$T_v = (T_a + Ä_t) + C_e/V_m + (C_e + (Q_{uanti}*C_v/V_{via}))$

3.5.3 Proposta do ambiente simulado

Esse estudo toma como objeto de estudo o direito de passagem dos veículos em cruzamentos, comumente conhecidos através do acionamento da luz verde do semáforo. Sendo este limitado ao estudo de cruzamentos em nível com vias de circulação em mão dupla.

Para isso é utilizado o simulador SACI (*Simple Agent Communication Infraestructure*) que é composto por um conjunto de agentes, onde estratégias de decisão são implementadas. Cada cruzamento é composto por quatro agentes que tem autonomia no controle do semáforo pertencente a sua célula. Além dos agentes-semáforo, o SMA é composto por um agente-simulador que é responsável por simular o mundo real. Esse agente viabiliza a percepção dos agentes-semáforo simulando valores dos sensores de passagem de veículos nas vias dos agentes-semáforo e o agente-simulador e informa a esses agentes os parâmetros de simulação por meio da linguagem XML (*eXtensible Markup Language*).

Essa interface também é responsável por alimentar o agente-simulador com as informações de frequência de veículos que passam pela via.

De forma resumida, após a formação dos agentes, o sistema trabalha da seguinte forma: os agentes-semáforos negociarão entre si o direito da passagem no cruzamento, comumente conhecido com a fase da luz verde. Sempre que um agente-semáforo identificar a necessidade do objeto de negociação (fase da luz verde) esse mandará uma requisição para o agente que possui o objeto e então esse agente decidirá se irá conceder a permissão para o agente solicitante ou não, baseado nas estratégias de decisões.

As respostas são enviadas ao simulador que as guarda em um arquivo de *log* para análise da simulação.

3.5.4 Agente semáforo

No cruzamento em nível e com fluxo interrompido, o gerenciamento de tráfego é geralmente feito com a alternância do direito de passagem dos veículos em suas vias. Sendo essa a função dos agentes-semáforo no ambiente simulado.

O agente-semáforo é composto por três módulos: a memória responsável pelo armazenamento do objeto de negociação e dos dados de decisões; modulo de decisão, responsável pela iniciativa das ações do agente; modulo de comunicação, composto pelos protocolos de comunicação e é responsável pela comunicação entre os agentes-semáforos e agente-simulador.

3.5.5 Módulo de memória

Esse módulo é responsável por armazenar as informações pré-estabelecidas necessárias para os cálculos dos tempos semafóricos como o tempo de atraso do primeiro veículo após a abertura do sinal, intervalo de tempo entre o arranque de dois veículos sucessivos, distância percorrida durante um período de embalagem e o comprimento médio de um veículo.

A memória também é responsável por armazenar algumas informações estáticas na vida do simulador como a identificação do semáforo (único no ambiente simulado), identificação dos semáforos ao redor (superior, direito, esquerdo e inferior), largura do cruzamento incluindo faixa de pedestres, velocidade permitida para a via e comprimento da via até a faixa de pedestres.

As informações dinâmicas passíveis de alteração durante a execução são: semáforo que possui o objeto de negociação; quantidade de carros que existem em sua via; tempo necessário para escoamento de sua fila; momento inicial em que um agente; passou o objeto para outro agente momento inicial do

recebimento do objeto pelo agente.

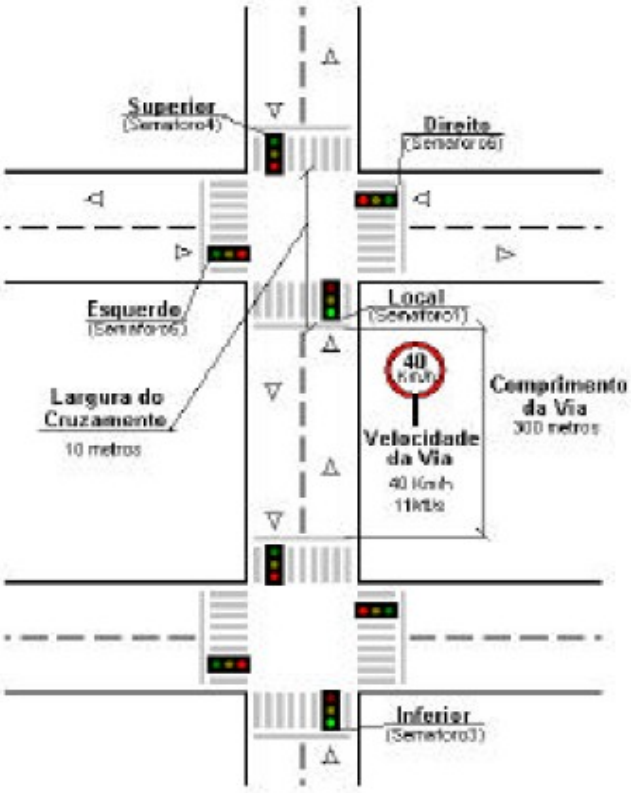

Figura 3.9 Variáveis do módulo de memória

3.5.6 Módulo de decisão

Dois algoritmos de decisão são implementados nos agentes-semáforo: o ativo que manda uma requisição para negociação do objeto e o passivo que envia uma mensagem de resposta a negociação do objeto.

A estratégia de decisão se apoia em dois fatores importantes que são a ocupação da via e a paciência do motorista com relação ao tempo levado para passar um cruzamento.

O fator ocupação é responsável por evitar o esgotamento da via com o excesso de veículos que esperam pela fase do verde.

O fator paciência é o fator responsável por limitar o tempo de espera de um motorista, evitando assim um período excessivo sem a negociação do objeto.

Alem disso, quando o objeto é negociado com sucesso, o agente terá que esperar pelo tempo do amarelo e a segurança do vermelho, antes de receber o objeto. Logo em seguida, o agente recém portador do objeto irá calcular o tempo necessário para evacuar a fila (tempo de verde) e armazenar o valor em *log* para futura utilização.

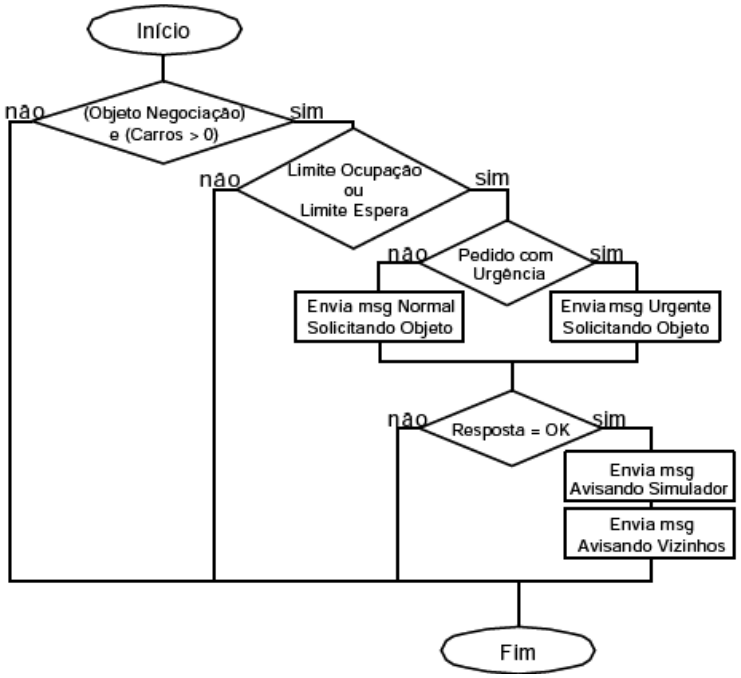

Figura 3.10 Decisão ativa

O agente-semáforo é capacitado também para, ao receber uma mensagem de pedido de negociação, decidir sobre a resposta que deverá

enviar. Para essas ocasiões, o fator de escoamento é configurado de modo a permitir um tempo mínimo de escoamento que o agente poderá utilizar.

3.5.7 Protocolo de comunicação entre os agentes

Uma vez que um agente tenha a necessidade de ter o objeto de negociação, esse agente manda uma mensagem pedindo o objeto ao agente que o possui. Esse agente então pode enviar uma resposta de aceitação ou rejeição.

No caso de aceitação, o agente enviará as informações dos tempos de segurança e somente depois desse período, o objeto será passado para o agente solicitante.

Uma vez que o agente solicitante receba a permissão esse informará o agente-simulador e os agentes-semáforos pertencentes ao mesmo cruzamento, sua situação atual, para que posteriormente os outros elementos possam identificar o portador do objeto.

3.5.7 Agente simulador

O agente simulador é geralmente criado após a criação dos agentes-semáforo. Ele tem como responsabilidade simular um ambiente real informando o volume de veículos que passam por segundo em cada via.

Outra função do agente-simulador é manter em arquivo de *log* as informações de todas as transações do objeto executadas pelos agentes-semáforo. Dessa forma, o arquivo pode ser posteriormente analisado.

3.5.8 Resultados

Para a análise dos resultados foram realizadas simulações em dezesseis diferentes malhas viárias, contendo até seis cruzamentos por malha e informando valores entre um e quatro carros por segundo como parâmetros de simulação.

Sendo utilizadas as estratégias de fator de ocupação e paciência verificou-se em mais de 60% dos casos que o objeto de negociação foi solicitado em função da ocupação das vias, não deixando assim os motoristas esperando por tempos excessivos.

Já os pedidos realizados em função do fator de paciência ocorreram em vias de baixo fluxo em função do tamanho da via.

Assim, a estratégia de decisão aplicada correspondeu às expectativas de não deixar os motoristas esperando por tempos maiores que os padrões permitidos e por não comprometerem as vias esgotando a capacidade de sua ocupação. Por outro lado, essa estratégia saiu do padrão de utilizar tempos de ciclos semafóricos fixos, onde comumente é dado o direito de passagem somente sobre a necessidade analisada pelo semáforo. Contudo, isso impediu o sincronismo de semáforos dispostos em série comumente conhecidos como “onda verde”.

4 Proposta para Controle do Tráfego Urbano

Devido a diversas peculiaridades em diferentes regiões, a complexidade na gerência do transito pode atingir diferentes níveis. Regiões com uma densidade demográfica superior tendem a encontrar problemas muitas vezes pela ausência de um planejamento condizente com o crescimento populacional.

O capítulo corrente traz uma alternativa possível orientada a baixo custo bem como a performance das decisões tomadas para a administração do transito. Esse abrange os métodos, a lógica e as ferramentas dentre os estudos anteriormente analisados que apresentam maior potencial para a administração da atividade, bem como inovações ao auferir erros em Redes Neurais de propagação reversa.

4.1 Aplicações e sistemas operacionais

O projeto foi desenvolvido de com base em software livre e de código aberto de acordo com os termos de licença GPL (*General Public License*). O objetivo com isso foi obter soluções de baixo custo bem como fazer uso de uma manutenção eficaz e de alta qualidade. Como sistema operacional base foi utilizado o *FreeBsd* versão 7.2 (FreeBSD Fundation. 2009), posteriormente o código foi portado para outros dois sistemas, o Slackware versão 12.1 (Volkerding, P. 1993) e o Suse versão 10 (Novell 1993. Suse).

As propriedades do *hardware* utilizado nos testes são PC portátil, processador Intel Centrino de 2.20GHz, memória RAM 4GB e disco rígido de 74GB.

A vasta flexibilidade quanto ao sistema operacional é dada ao fato da portabilidade do compilador *gcc* que por sua vez foi utilizado para compilar os

códigos fonte.

Para o desenvolvimento das Redes Neurais foi utilizada a biblioteca *Fast Artificial Neural Network* (Nissen, S. 2009) desenvolvida na linguagem de programação C. Partindo desse pressuposto foi adotada a linguagem C para o projeto como um todo.

4.2 Os dados de entrada

A proposta do estudo é uma solução alternativa que combina duas metodologias algorítmicas. Os algoritmos de propagação reversa que tem como objetivo identificar uma função e a partir dela classificar a informação de entrada e os Algoritmos Genéticos que trabalham com as Redes Neurais de forma a definir novas redes a partir de cruzamentos genéticos.

Atualmente, o controle de transito em sua maioria é feita com base no tempo de abertura para cada fluxo. Quando há a necessidade de mudanças dos tempos pré-definidos, seja por conta de períodos de pico ou imprevistos como acidentes ou manutenções, essas são geralmente executadas manualmente.

O modelo numérico foi desenvolvido através da analise de possibilidades em um ambiente real. Dessa forma foi definida uma saída esperada para cada possível entrada, esses dados são utilizados durante a faze de treinamento da rede.

As informações de entrada estão armazenadas em um arquivo chamado *traffic.data* e pode ser encontrado no diretório *tdata*. Esse arquivo é estruturado da seguinte forma:

- A primeira linha contem três parâmetros sendo o primeiro a quantidade de pares de treinamentos presentes no arquivo, o segundo o número de neurônios na camada de entrada, no caso desse estudo

são quinze e o terceiro parâmetro representa o número de neurônios na camada de saídas;

Ex.:

160 15 5

As camadas de entrada das Redes Neurais são compostas por quinze neurônios cada e devem ser analisadas da seguinte forma:

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

- Os sete valores binários que compõe o bloco mais significativo representam o volume de trafego de uma das vias;
- Os sete valores seguintes representam o volume de trafego da outra via;
- E o último neurônio menos significativo é utilizado para indicar a rede que o que tempo de espera máximo foi atingido (Weiss 1999 e Bordini 2001).

As camadas de saída das Redes Neurais são compostas por cinco neurônios cada:

0 0 0 0 0

- O neurônio que compõe a saída mais significativa é utilizado para priorizar um dos fluxos de trafego, isto é, quando estiver ativo a saída prioriza o fluxo referente aos sete neurônios de entrada mais significativos, quando estiver desativo a saída prioriza o fluxo dos sete neurônios menos significativos da camada de entrada.
- O restante dos neurônios definem o tempo que o semáforo deve permanecer em determinado estado para que permita a vazão do trafego.

Os valores das camadas de entrada e saída intercalam entre si até o fim

do arquivo formando os pares de treinamento.

4.3 A Rede Neural e a Estrutura Genética

O modelo é composto por um algorítimo de propagação reversa, descrito em detalhes no **capitulo 2,** que deve ser analisado como a primeira etapa do processo. E uma segunda etapa composta por um algoritmo de evolução genética que tem como principal função selecionar dentre as redes geradas, as que apresentam a maior habilidade genética ou maior capacidade de acertos nas decisões para o controle do tráfego.

Aquém da seleção dos indivíduos, redes de maior aptidão, o Algoritmo Genético tem outra tarefa de suma importância que é estabelecer relacionamentos entre os indivíduos de uma população. A população tem uma densidade limitada e pré-definida. Os novos indivíduos gerados de forma aleatória tendem a ter resultados iguais ou superiores quanto a classificação dos fluxos de trânsito se comparados a seus geradores.

4.4 Analise do código

Para facilitar a analise do código ele foi particionado em três secções e para cada secção existe um *use case*:

- Criação e treinamento das Redes Neurais;
- Estrutura de registros;
- Algoritmo Genético;

Essas secções interagem entre si de forma a criar, selecionar e multiplicar as Redes Neurais do sistema.

Todos os detalhes do código fonte estão disponíveis para analise no **Apêndice C.**

4.4.1 Criação e treinamento das Redes Neurais

A secção que define a criação e o treinamento das redes tem constantes pré definidas que são utilizadas para estruturar as redes. A primeira fase é declaracão dessas constantes que são:

- a constante num_layers que define a quantidade de camadas;
- num_input o número de neurônios na camada de entrada;
- num_hidden0 o número de neurônios na primeira camada oculta;
- num_hidden1 o número de neurônios na segunda camada oculta;
- num_output o número de neurônios na camada de saída.

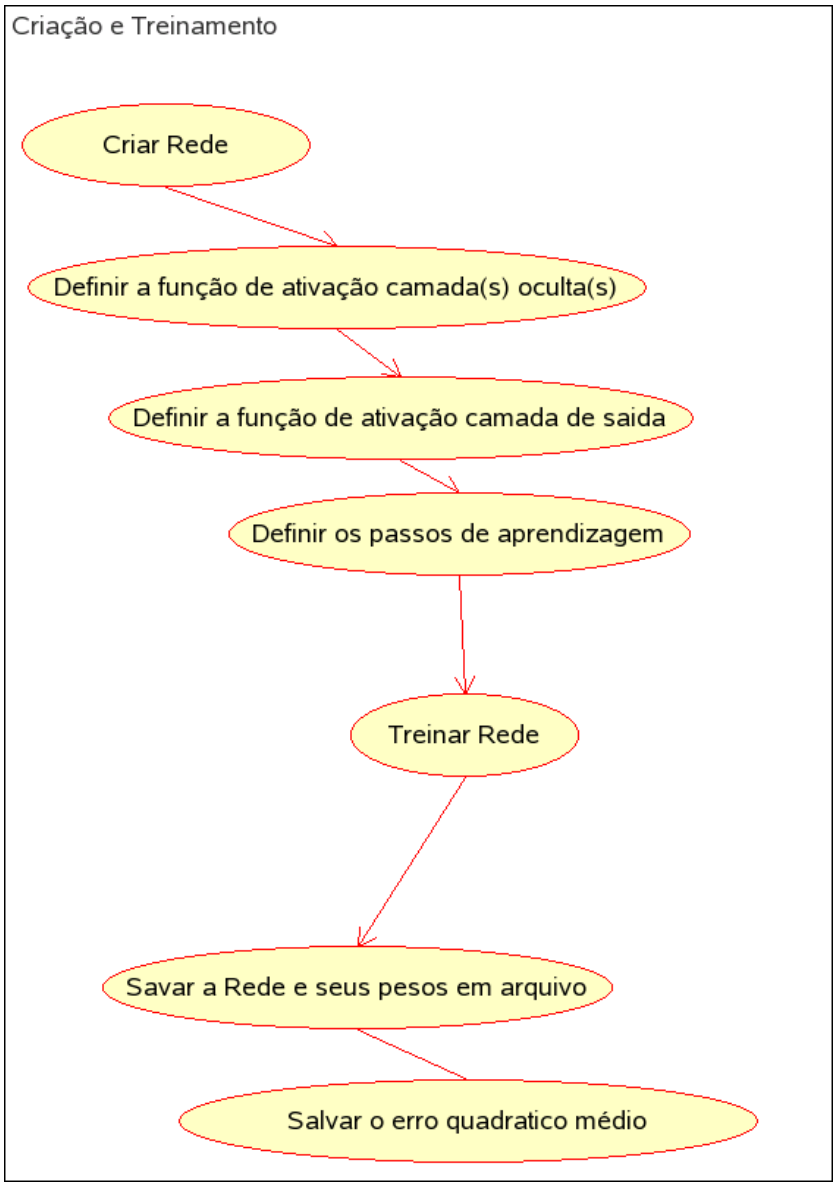

Figura 4.1 *Use case* ilustra a criação e o treinamento das Redes Neurais

Os valores das constantes que definem o número de neurônio na entrada e na saída foram criados de tal forma a atender diferentes senários desde os

mais simples aos mais complexos. Experimentos foram feitos com diferentes valores para as camadas ocultas e seus neurônios. Quando se tem uma quantidade maior de neurônios que a utilizada para os dados de entrada, as redes levam mais tempo na faze de treinamento e quando se tem menos neurônios existe a possibilidade das redes não chegarem ao erro esperado. Todos esses valores podem ser alterados caso haja a necessidade na resolução de um problema.

A função de ativação utilizada nas redes é a função senoidal, como justificado no **capítulo 2** essa é a função que melhor se destaca dentre as outras para a classificação de padrões.

Seguindo a declaracão das constantes existem algumas etapas que vão desde a criação das redes até o treinamento e armazenamento das informações para uso futuro como mostra a **Figura 4.1**.

A Rede Neural é criada com informações pré-definidas na biblioteca FANN (Nissen, S. 2009), isso ocorre durante a execução da ação Criar Rede. Assim faz-se necessária a definições de alguns parâmetros para que a rede, antes de ser treinada, tenha as características esperadas pelo projeto. Nesse caso essas características são definidas com as acoes:

- Definir a função de ativação na(s) camada(s) oculta(s);
- Definir a função de ativação na camada de saída;
- Definir os passos de aprendizagem[2].

Com esses parâmetros definidos, a rede pode ser treinada e posteriormente ter suas informações armazenadas em seu respectivo registro bem como seu erro quadrático médio. Essas informações são utilizadas futuramente tanto para a execução da Rede Neural quando para a analise genética.

2 Os passos de aprendizagem têm o mesmo significado de deslocamento do aprendizado definido no capitulo 2.1.2

O erro quadrático médio esperado é especificado no momento da execução da função de treinamento, caso a rede obtenha um erro inferior ao definido o treinamento é cessado. Caso contrario o treinamento continua até que o número máximo de *epochs*[3] seja atingido.

4.4.2 Estrutura de registros

Como citado no item anterior, após a execução do treinamento o erro é salvo em um registro. Esses registros são representados no *use case* como registro de erro e, após o período de treinamento, cada Rede obtém o seu registro de erro.

As informações para o treinamento das Redes Neurais estão armazenadas em um conjunto de registros chamado registros de pares de treinamento. As Redes são treinadas com base nas informações contidas nesse registro. A estrutura de registros é ilustrada na **Figura 4.1**.

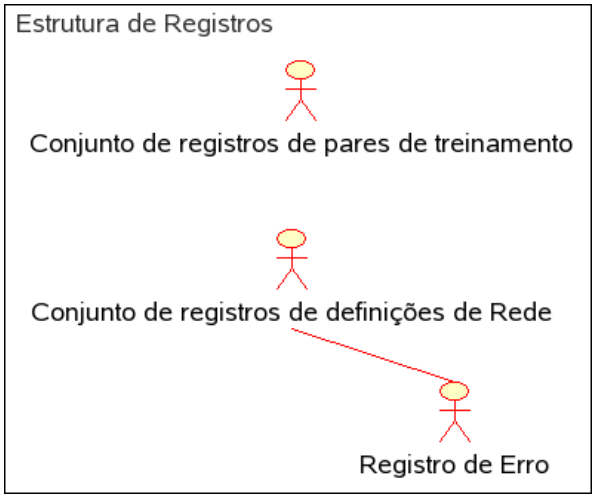

Figura 4.2 *Use case* ilustra a estrutura de registros

3 Durante um *epoch* cada par de treinamento é utilizado para uma iteração.

Além do erro, após o treinamento de cada Rede, são armazenados os pesos calculados para cada conexão bem como informações de onde cada neurônio está conectado. Posteriormente cada rede pode ser criada individualmente ou em grupos de acordo com a necessidade do projeto por meio do conjunto de registros de definições de Rede.

4.4.3 Algoritmo Genético

O algoritmo evolucionário genético é associado a solução com o objetivo de aumentar a precisão das Redes Neurais. Para isso esse algoritmo reproduz Redes filhas definindo seus pesos iniciais com os valores que geraram os menores erros nas conexões de seus pais em treinamentos anteriores.

A implementação do algoritmo genético é baseado na metodologia parcial descrita em detalhes no **capitulo 2**. O algoritmo é composto por duas etapas:

- Ordenar Redes por erros;
- Executar Cruzamentos.

Conforme ilustração no *use case* **Figura 4.3**.

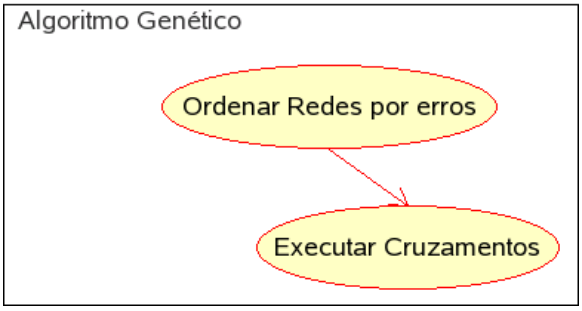

Figura 4.3 *Use case* ilustra o Algoritmo Genético

A ação Ordenar Redes por erros ordena as Redes em ordem decrescente e registra os valores de seus índices e respectivos erros. Esses valores são armazenados em um *array* e ficam disponíveis para utilização durante a execução.

A ação Executar Cruzamentos é uma iteração onde redes filhas são criadas a partir do cruzamento de duas Redes. Esse processo é executado entre todas as Redes. Como parte dele cada conexão entre cada neurônio é analisada . O indivíduo que tiver o melhor erro para uma conexão em especial cede o valor do peso utilizado para seu herdeiro gerando dessa forma uma Rede com as melhores características de seus sucessores.

Cada nova rede criada passa pelo processo de treinamento porém o treinamento é realizado com um passo de aprendizagem inferior ao utilizado no treinamento das redes sucessoras. O motivo é aumentar a precisão da função de aprendizagem reduzindo o erro médio quadrático final.

4.4.4 O conjunto como um todo

O conjunto como um todo ilustra como os blocos descritos no **capítulo 4** exercem uma interação.

No *use case* da **Figura 4.4** aparecem dois atores:

- Entidade de controle;
- Rede Neural.

A entidade de controle é responsável por iniciar os processos de criação e treinamento das redes e o processo do algoritmo genético. Ambos processos não devem ser iniciados simultaneamente. O motivo é que a quantidade de novas redes somente é registrada quando o processo de treinamento é

finalizado, assim o processo genético, caso seja executado antes da finalização do treinamento, considera apenas as redes antigas e não as que estão no processo atualmente.

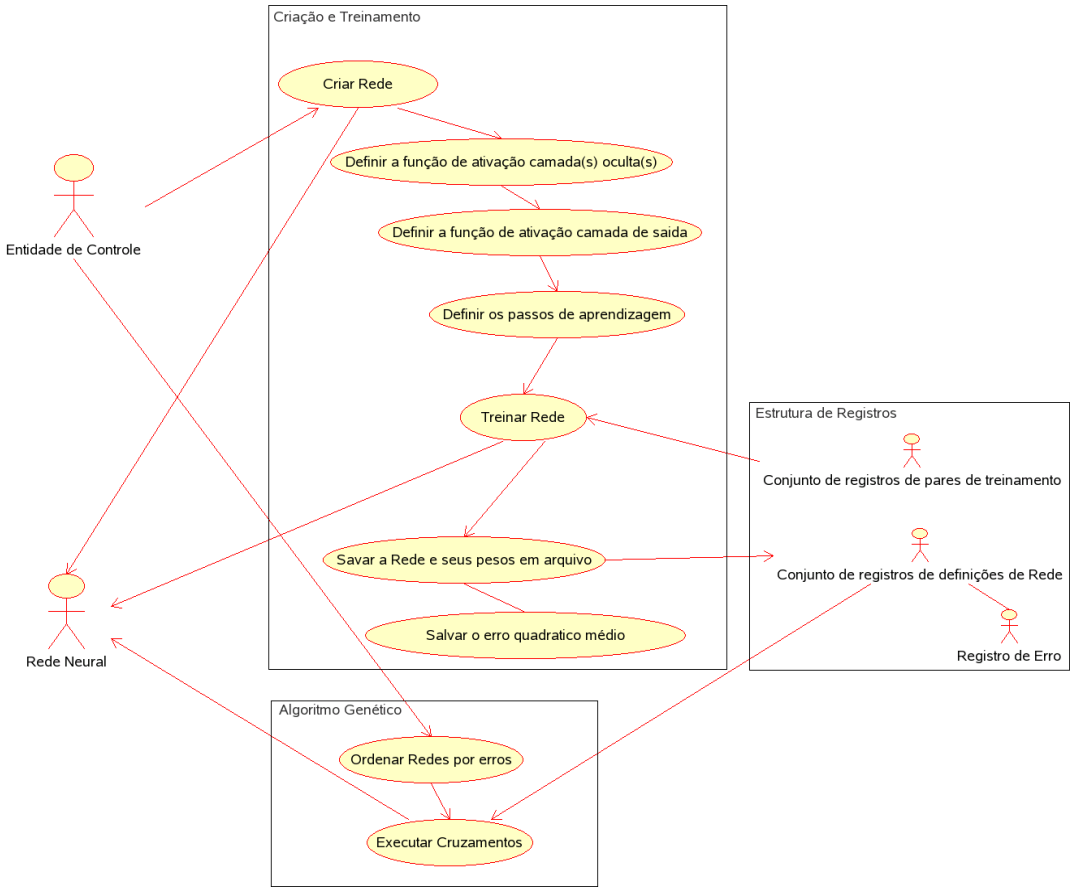

Figura 4.4 *Use case* ilustra a interação do conjunto

O segundo ator é a Rede Neural propriamente dita. Esse ator representa cada Rede desde o momento em que ela é criada até o momento em que ela é destruída pelo programa após ser salva em registros, isto é, o momento em que a Rede está ativa em memória volátil e, treinada ou não, pode receber dados de entrada.

A ação criar Rede inicializa uma nova Rede Neural em memória, essa rede passa pelo processo de aprendizagem, seus registros são salvos e finalmente o espaço em memória utilizado para processa-la é liberado para a próxima rede.

5 Resultados

Os processos foram executados utilizando como base registros de treinamentos com volumes de fluxos de trafego. Diversas situações com diferentes volumes de trafego são simuladas por esses registros.

Gráficos foram extraídos para facilitar a visualização do treinamento das redes. Os gráficos mostram os valores dos erros e seus respectivos pesos para cada conexão de uma Rede em especifico. Cinco Redes Neurais foram geradas para a analise nesse estudo. Essas Redes poderiam gerenciar cinco cruzamentos com semáforos. Após a criação e treinamento das cinco Redes é executado o processo genético e esse gera mais vinte Redes filhas que serão analisadas posteriormente. O sistema como um todo conta com 25 Redes para analise do corrente estudo não limitando-se a esse número.

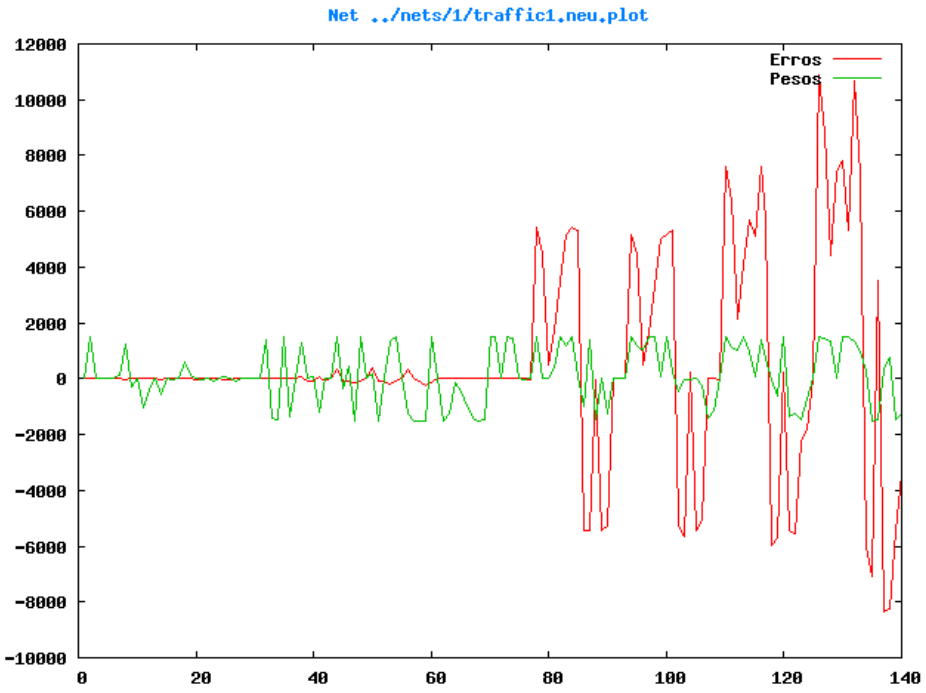

Figura 5.1 Gráfico comparando pesos e erros da primeira Rede

A Figura 5.1 ilustra a primeira Rede Neural após o treinamento por propagação reversa, nota-se que a Rede tem um índice alto de erros em cerca de 45% de suas conexões. Nesse estágio o algoritmo genético ainda não foi aplicado. Esta é uma rede candidata a melhorias após os cruzamentos genéticos.

A segunda Rede Neural, ilustrada na **Figura 5.2**, também apresenta erros com valores significativos em aproximadamente 45% de suas conexões. Porém os picos de erro não ultrapassam a faixa dos 5000 enquanto que na primeira rede os picos chegam à aproximadamente 11000.

Uma característica importante é que as Redes neurais tem seus pesos gerados de forma aleatória, por esse motivo elas podem apresentar diferentes características mesmo após o treinamento.

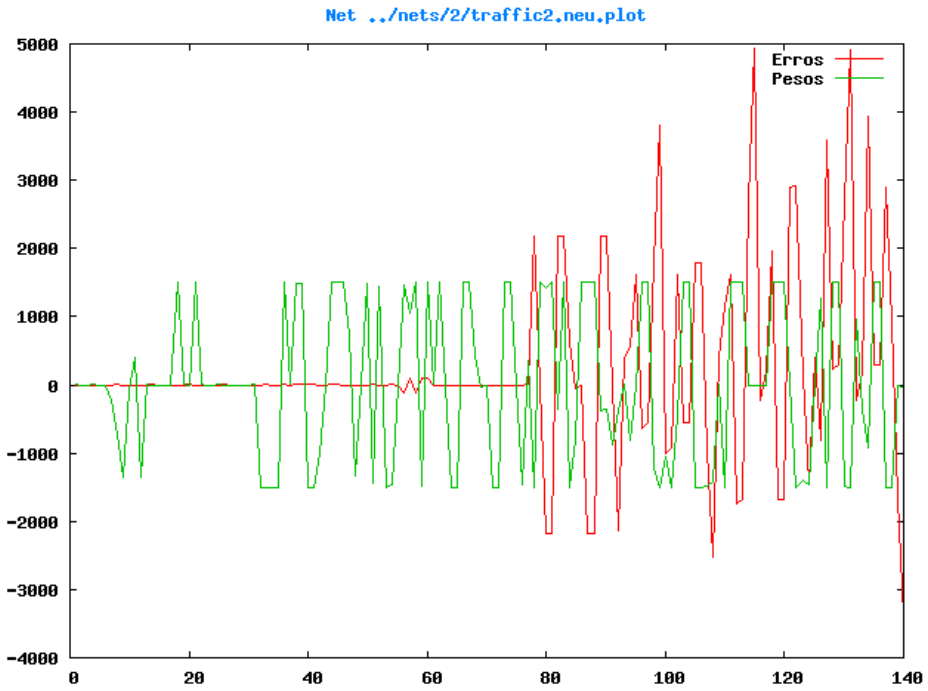

Figura 5.2 Gráfico comparando pesos e erros da segunda Rede

A terceira Rede, **Figura 5.3** tem um comportamento diferente das anteriores. Nesse caso o algorítimo de propagação reversa teve um bom desempenho durante o treinamento e seu produto poderia ser utilizado para gerenciar os fluxo de volumes de semáforos mesmo antes de passar pelo algoritmo genético.

O algoritmo genético dificilmente provê melhorias para redes que seguem o perfil da terceira Rede.

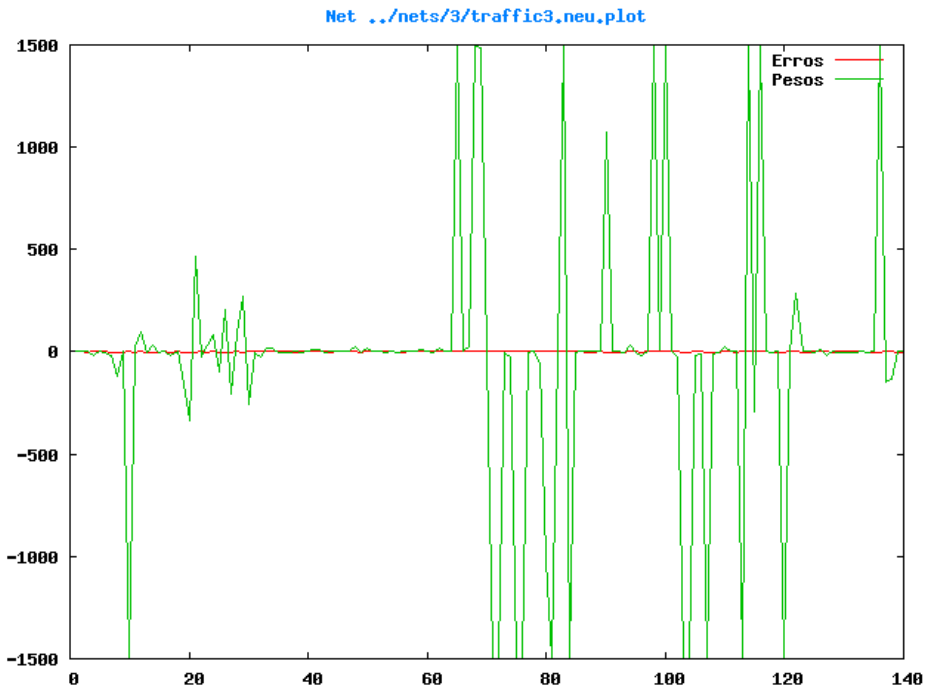

Figura 5.3 Gráfico comparando pesos e erros da terceira Rede

A quarta Rede tem um perfil semelhante ao da terceira e também poderia ser utilizada para gerenciar fluxos de volumes de semáforos mesmo antes da

analise genética. Conforme ilustração na **Figura 5.4** nota-se que tanto os valores dos pesos quanto os valores dos erros estão próximos aos da terceira Rede.

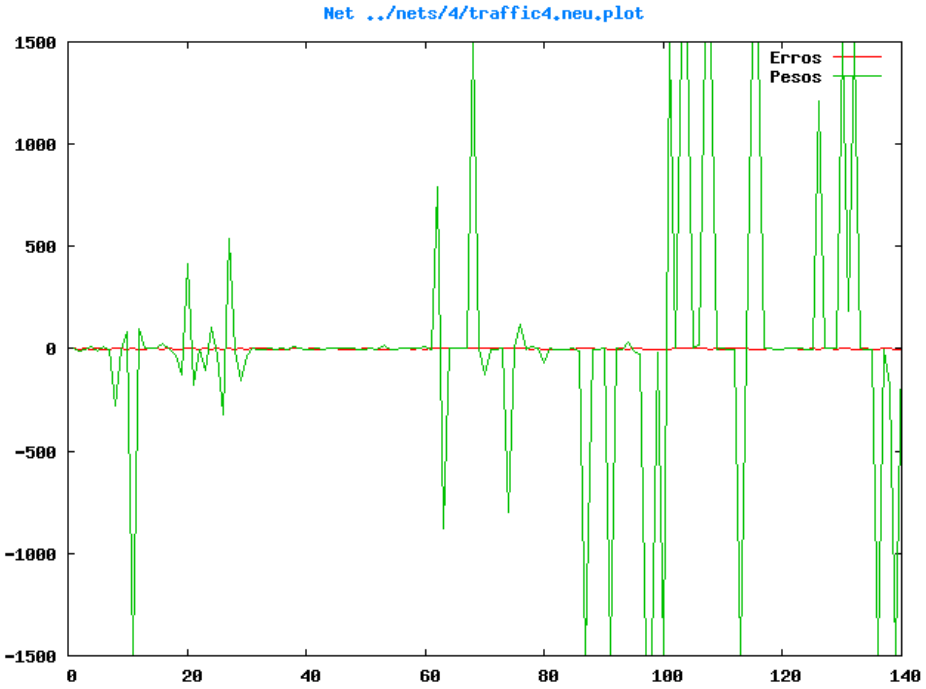

Figura 5.4 Gráfico comparando pesos e erros da quarta Rede

A quinta Rede apresenta características diferentes de todas as outras. Ela tem valores de erros superiores aos valores da primeira Rede. Uma rede como essa seria inviável para o gerenciamento de fluxos entretanto é uma rede de características interessantes a serem tratadas pelo algoritmo genético. Ela atinge valores de erros aproximados à 30000.

Essa rede é a ultima a ser analisada antes do processo genético. Como mostra a **Figura 5.5** é uma rede passível de melhoria e ajuste de seus pesos em busca de erros mais baixos.

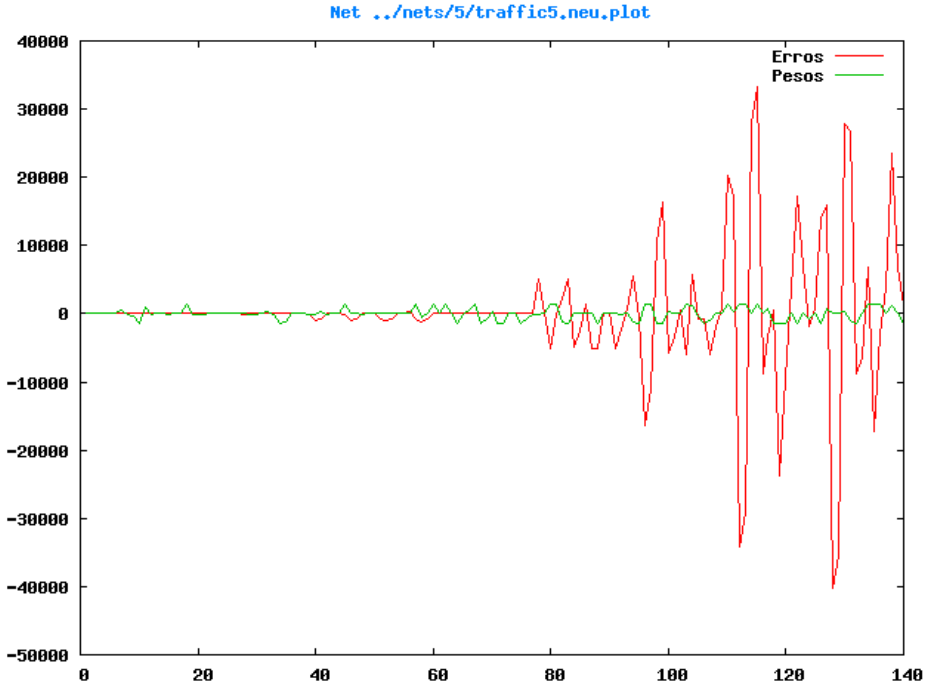

Figura 5.5 Gráfico comparando pesos e erros da quinta Rede

Após o processo de criação e treinamento, é executado o processo genético como descrito no *use case* interação do conjunto no **capítulo 4**. Nessa fase as redes passam por cruzamentos onde cada Rede Neural é combinada com outra para gerar uma Rede Neural filha. A Rede filha tem as mesmas propriedades de suas sucessoras. A diferença é que os pesos são copiados da sucessora que tem o menor erro para aquela determinada conexão. Assim os cruzamentos tendem a trazer resultados com erros menores aos das Redes sucessoras. Porém quando as redes sucessoras já apresentam baixos valores de erros dificilmente o algoritmo genético os pode reduzir.

Os cruzamentos são feitos de forma a cada rede se relacionar com todas as outras exceto com ela mesma. Neste caso a primeira rede se relaciona com a segunda, terceira, quarta e quinta. A segunda se relaciona com a primeira,

terceira, quarta e quinta e assim sucessivamente.

Dado o fato da grande quantidade de Redes geradas pelo algoritmo genético (vinte) a analise dessas será feita a partir da rede que apresentar a maior performance (baixos erros). As redes restantes estão disponíveis no **Apêndice E** para maiores analises.

Dentre as novas redes geradas a partir do cruzamento da primeira Rede com as demais, a que apresenta os menores erros é a que tem como sucessoras a primeira e a segunda rede. Como ilustrado na **Figura 5.6** a curva de erros tem valores consideravelmente inferiores aos da primeira. Essa nova Rede apresenta apenas um pico de aproximadamente 9000 enquanto a primeira rede apresentou múltiplos picos de 11000. Picos de 8000 foram reduzidos a 5000 ou menos e picos de 6000 reduzidos a 4000. Além da área de erros estar em aproximadamente 35% ao invés dos 45% apresentados na primeira Rede.

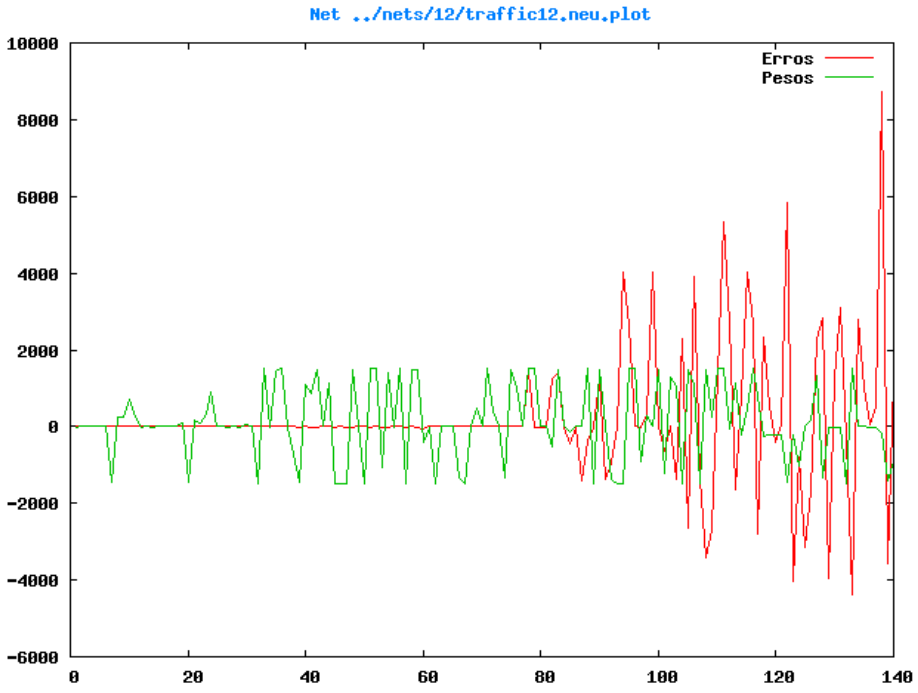

Figura 5.6 Gráfico de pesos e erros no cruzamento 1x2

Os cruzamentos realizados pela segunda rede com as outras apresentou uma melhoria de maior grandeza se comparado com a Rede anterior. O cruzamento que obteve os menores erros foi entre a segunda e a primeira.

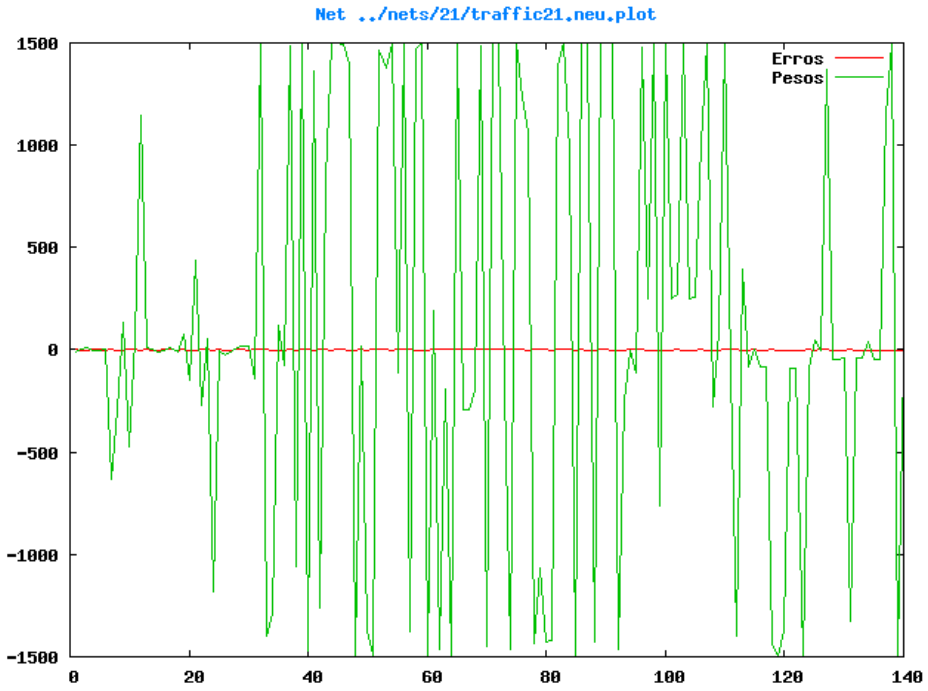

Figura 5.7 Gráfico de pesos e erros no cruzamento 2x1

Um fato de grande importância é que essa rede filha poderia substituir ambas sucessoras isto é a segunda e a primeira uma vez que esta obteve melhores resultados em comparação a Rede filha anterior e suas sucessoras. Seus erros estão próximos de zero como ilustra a **Figura 5.7**.

A terceira rede, como citado anteriormente, obteve baixos valores de erros apenas com o treinamento proposto pelo algoritmo de propagação reversa. Neste caso o algoritmo genético dificilmente chega a uma Rede com erros inferiores. De qualquer forma os cruzamentos foram executados e duas Redes

foram geradas com qualidades compatíveis as da terceira Rede.

São elas as Redes Neurais do cruzamento de 3x2 e 3x4. O cruzamento de 3x2 é ilustrado na **Figura 5.8** e o restante dos cruzamentos podem ser encontrados no **Apêndice E**.

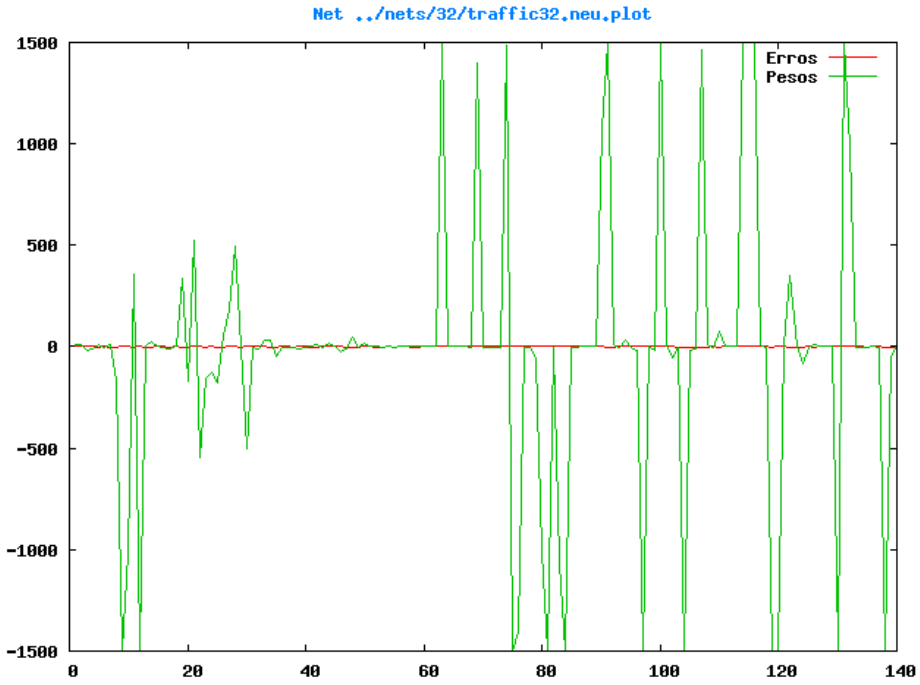

Figura 5.8 Gráfico de pesos e erros no cruzamento 3x2

Nota-se que a Rede apresenta um erro constante muito próximo de zero como sua sucessora a terceira Rede. Porém um fato importante é que a segunda rede, também sucessora desta Rede filha, poderia ser substituída por esse produto com erros inferiores a ela.

A quarta Rede segue o mesmo perfil da terceira e obtém como melhor Rede filha o cruzamento entre 4x1. Essa Rede tem erros constantes e próximos de zero como no caso anterior e também é uma opção para a substituição da primeira rede.

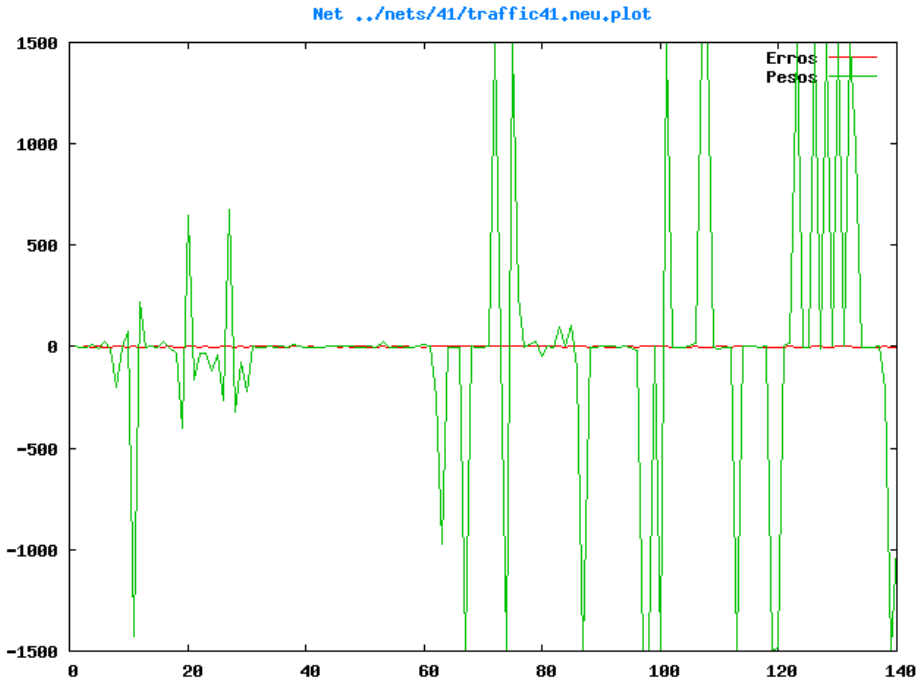

Figura 5.9 Gráfico de pesos e erros no cruzamento 4x1

A quinta Rede tem como melhor Rede filha o cruzamento entre 5x4. A Rede filha gerada sofre mudanças intensas quanto a seus erros obtendo reduções de ~30000 para ~4000. Embora haja a presença de erros na Rede, esses tem grandezas menores e essa Rede apresenta aptidão para gerenciar fluxos de volumes de trafego.

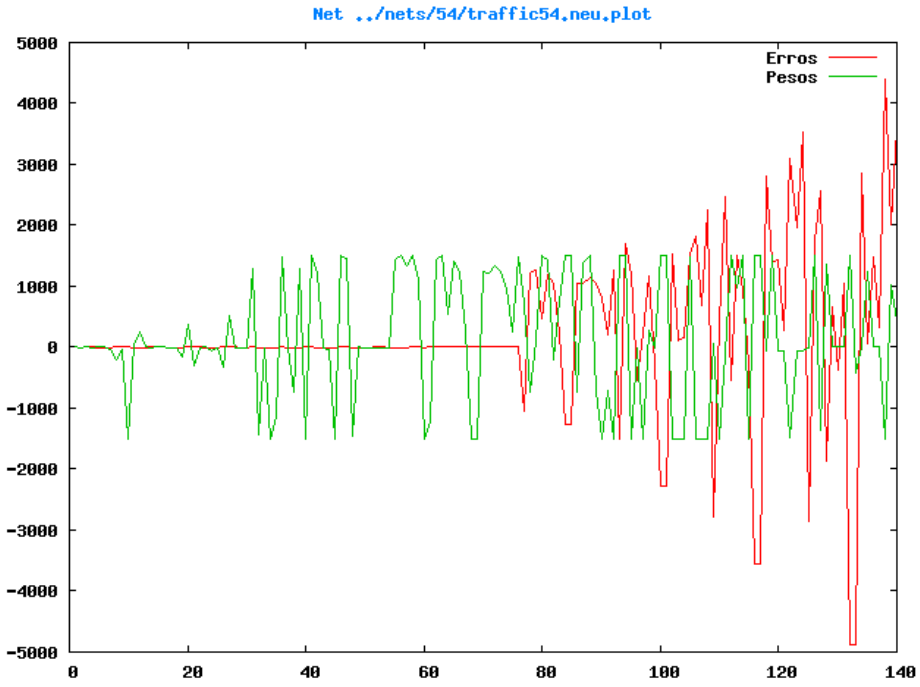

Figura 5.10 Gráfico de pesos e erros no cruzamento 5x1

Como complemento as curvas ilustradas no **capítulo 5** foram extraídos erros quadráticos médios de todas as Redes no instante após o treinamento. Os erros, Inclusive os das Redes filhas, estão disponíveis na **Tabela 5.1**. Essa tabela mostra de uma forma simplificada o que as curvas de erro mostraram até então em maiores detalhes.

O erro quadrático médio mostra o comportamento da Rede e sua capacidade de aprendizagem durante o período de treinamento. A analise desses erros torna possível uma classificação das Redes por sua capacidade de gerenciar os fluxos de trafego.

Redes Neurais Primarias	Redes Filhas Cruzamentos	Erro Quadrático Médio
1		0.10521
	1x2	0.03636
	1x3	0.04737
	1x4	0.07131
	1x5	0.11382
2		0.09715
	2x1	0.04794
	2x3	0.08631
	2x4	0.17530
	2x5	0.16184
3		0.00490
	3x1	0.10707
	3x2	0.00490
	3x4	0.00490
	3x5	0.00490
4		0.00490
	4x1	0.00490
	4x2	0.22167
	4x3	0.09586
	4x5	0.07121
5		0.14596
	5x1	0.23307
	5x2	0.16000
	5x3	0.09482
	5x4	0.07198

Tabela 5.1 Comparação dos erros quadráticos médios

Como exemplo na Tabela 5.1 tem-se para a primeira rede um erro de 0.10521 e nos cruzamentos encontra-se o menor erro que foi gerado pela Rede 1x2 proveniente do cruzamento da primeira com a segunda rede e é de 0.03636.

A mesma analise foi feita para as demais redes.

Após essa primeira analise seletiva faz-se uma analise minuciosa dos erros entre as conexões existentes em Redes específicas como feito no início do capítulo corrente por meio de analise gráfica.

A análise em detalhes das conexões e os cruzamentos genéticos foram de grande importância para o projeto pois dessa forma foi possível identificar os erros e corrigi-los modificando os pesos por valores que obtiveram um melhor retorno em outras redes. Geoffrey Hinton comenta, em uma palestra para o Google (Hinton, G. 2007), que existe a necessidade de identificar as informações produzidas pelos neurônios para que seja possível a correção de seus erros posteriormente. A implementação do algoritmo genético requer o mapeamento dos erros, isto é, essa implementação é capaz de identificar e alterar pesos de acordo com os erros em cada conexão.

As novas Redes, criadas a partir de cruzamentos genéticos, tem seus pesos mapeados e manipulados com o objetivo de obter menores erros.

O objetivo foi alcançado ao obter erros tão baixos quanto ou inferiores aos dos estudos analisados no **capítulo 3**. Bem como mostrado neste capítulo a implementação híbrida aumentou ou manteve a eficiência do convencional algoritmo de propagação reversa em 100% dos casos.

5.1 Estudos futuros

Estudos futuros podem introduzir maiores possibilidades para o algoritmo híbrido e sua implementação. Algumas sugestões para estudos futuros dando continuidade a ideologia híbrida são:

- Analise para implementação de semáforos para ciclistas e pedestres;
- Inserir características no algoritmo genético como limite populacional,

diversidade e outras citadas no **capítulo 2**;

- A aplicação FANN (Nissen, S. 2009), por ser de código aberto, pode ser alterada para conter códigos genéticos;
- Estudo e implementação de *labels* na aplicação FANN;
- Estratégias para a construção civis de vias constantes, evitar reduções de faixas ou interrupções de vias.

Um trabalho como esse pode envolver muitos aspectos e abrir uma ampla possibilidade de estudos por tratar de um assunto complexo e muitas vezes problemático em todo o mundo.

Referências Bibliográficas

Abarbanel, H. D. I. 1996. *Analysis of Observed Chaotic Data*, Springer Verlag, New York, NY.

Annunziato, M. e Abarbanel, H. D. I. 1999. *Non Linear Dynamics for Classification of Multiphase Flow Regimes, Proc. Of Int. Conf. Soft Computing*, SOCO, Genoa, Italy.

Annunziato, M. e Pizzuti, S. 2002. *Adaptive Parameterisation of Evolutionary Algorithms and Chaotic Population Dynamics,* International Journal of Knowledge-Based Intelligent Engineering System, 170-177.

Bordini, R. H., Vieira, R., Moreira, A. F. 2001. *Fundamentos de sistemas multiagentes*: XX Jornada de atualização em informática (JAI), volume 2, capítulo 1, 3 44p. SBC, Fortaleza, CE, Brasil.

Box, G. e Jenkins, J. 1970. *Time Series Analysis:* Forecasting and Control. San Francisco, Holden-Day .

Ejzemberg, S. 1996. *Análise da circulação e fluxos de tráfego*. São Paulo: Companhia de Engenharia de Tráfego.

Espel, M. 2000. *O controle eficaz dos semáforos para melhoria do tráfego* urbano. 53f. Monografia, Universidade Católica de Santos, Santos.

George F. Luger. 2002. *Artificial Intelligence:* Structures and Strategies for complex Problem Solving. 4ª Edição. Addison-Wesley.

Hebb, D. O. 1949. *The Organization of Behavior*. Nova York: Wiley.

Koutsopoulos, H. e Xu, H. 1992. *An Information Discounting Routing Strategy for Advanced Traveler Information Systems, Transportation research Part C,* Vol. 1, No.3, 249-264.

Lingras, P. J., Sharma, S. C. e Zhong, M. 2002. *Prediction of Recreational Travel using Genetically Designed Regression and Time Delay Neural Network Models*. Transportation Research Record 1805, Transportation Research

Board, Washington, DC: 16-24.

Lopes, M. A. R. 1998. *Código de trânsito brasileiro anotado*. São Paulo, SP: Ed. Revista dos Tribunais.

Minsky, M e Papert, S. 1969. *Perceptrons*: An Introduction to Computational Geometry. Cambridge, MA: MIT Press.

McCulloch, W. S. e Pitts, W. 1943. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*.

Nilsson, N. J. 1965. *Learning Machines*. New York, NY: McGraw-Hill.

Petrowski, A., Dreyfus, G. e Girault, C. 1993. Performance analysis of a pipelined backpropagation parallel algorithm. *IEEE Transations on Neural Networks,* 4(6):970-981.

Quinlan, P. 1991. *Connectionism and Psychology*. Chicago, IL: University of Chicago Press.

Rochester, N., Holland, J. H., Haibit, L. H. e Duda, W. L. 1988. Test on a cell assembly theory of the actuation of the brain, using a large digital computer. Em J. A. Anderson e E. Rosenfeld, ed. *Neurocomputing:* Foundations of Research. Cambridge, MA: MIT Press.

Stephanedes, Y. J., Michalopoulos, P. G. e Plum, R. A. 1981. Improved Estimation of Traffic Flow for Real time Control. *Transportation Research Record 795,* Transportation Research Board, Washington, D.C., 28-39.

Stern, Y et al. 1969. *Um estudo sobre tráfego:* sincronização de sinais. Rio de Janeiro, RJ: Instituto de Pesquisas Rodoviárias.

Vilanova, Luis Molist. 1985. Dimensionamento do tempo de amarelo. Nota técnica 108. São Paulo: Companhia de Engenharia de Tráfego. 16p.

Weiss, G. 1999. *Multiagent systems:* A modern approach to distributed artificial intelligence. Massachussetts: MIT Press.

Yoshizaki, A. 1995. Methodology of vehicle detector installation: arrangements in

an urban traffic control system. Proceedings of the Second World Congress on Intelligent Transport System, Yokohama, Japan.

Web grafia

Annunziato, M., Bertini, I., Pannicelli, A., Pizzuti, S. 2003. *Evolutionary feed-forward neural networks for traffic prediction*. Disponível em: <http://home.tele2.it/stefanopizzuti/papers/eurogen03_129.pdf>. Acessado em: 11/03/2008.

Annunziato, M. e Pizzuti, S. 2004. *A smart-adaptative-system based on evolutionary computation and neural networks for the on-line short-term urban traffic prediction*. Disponível em: <http://home.tele2.it/stefanopizzuti/papers/13650_P_Pizzuti.pdf>. Acessado em: 11/03/2008.

Fotouhi, F., Hwang, D., Xu, C. Z. 2005. *A parallel backpropagation learning algorithm for urban traffic congestion measurement*. Disponível em: <http://www.ece.eng.wayne.edu/~czxu/paper/annie99.ps>. Acessado em: 13/02/2008.

FreeBSD Fundation 2009. FreeBSD. Disponível em: <http://www.freebsd.org/>. Acessado em: 29/06/2009.

Hinton, G. 2007. The Next Generation Of Neural Networks. Disponível em: <http://www.youtube.com/watch?v=AyzOUbkUf3M>

Hübner, J. F. e Schimitz, M. 2002. Uso de SMA para avaliar estratégias de decisão no controle de tráfego urbano. Disponível em: <http://www.inf.furb.br/~jomi/pubs/2002/Schmitz-seminco2002.pdf>. Acessado em: 02/05/2007.

Lingras, P. J., Sharma, S. C. e Zhong, M. 2006. *Genetically-Designed Time Delay Neural Networks for Multiple-interval Urban Freeway Traffic Flow Forecasting*. Disponível em: <http://v8nu74s71s31g374r7ssn017uloss3c1vr3s.unbf.ca/~ming/document/neural-freeway.pdf>. Acessado em: 11/03/2008.

Nissen, S. 2009. Fast Artificial Neural Network Library (FANN). Disponível em: http://leenissen.dk/fann/ . Acessado em: 12/10/2008.

Novell 1993. Suse. Disponível em: <http://www.novell.com/linux/>. Acessado em: 29/06/2009.

Volkerding, P. 1993. Slackware. Disponível em: <http://www.slackware.org/>. Acessado em: 29/06/2009.

Wikipedia. 2009. John Von Neumann. Disponível em: <http://en.wikipedia.org/wiki/John_von_Neumann>. Acessado em: 29/01/2009.

Apêndice A – Histórico das Redes Neurais

O exemplo de computação neural mais antigo é o neurônio de McCulloch-Pitts (McCulloch e Pitts 1943). As entradas possíveis para o neurônio de McCulloch-Pitts são excitatórias (+1) ou inibitórias (-1). A função de ativação multiplica cada entrada pelo seu peso correspondente e soma os resultados; se a soma é maior ou igual a zero, o neurônio retorna +1, caso contrario, o retorno será -1. McCulloch e Pitts mostraram como esses neurônios poderiam ser construídos para computar qualquer função lógica, demonstrando que sistemas criados com esses neurônios disponibilizam um modelo computacional completo.

A **Figura A.1** mostra os neurônios de McCulloch-Pitts para calcular funções lógicas. O neurônio que calcula a função E tem três entradas: **x** e **y** são os valores a serem analisados; o terceiro, algumas vezes chamado de polarizador, tem um valor constante de +1. As entradas de dados e o valor polarizador tem os pesos de +1, +1, e -2, respectivamente. Assim para qualquer valor de **x** e **y**, o neurônio calcula **x** + **y** – 2: se este valor é menor que 0, ele retorna -1, senão +1.

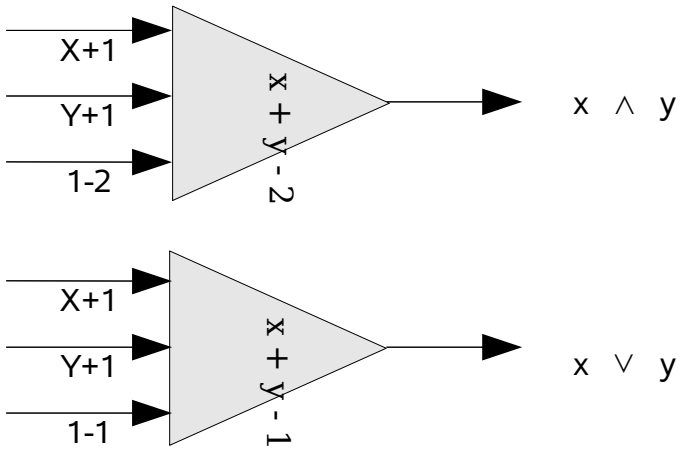

Figura A.1 McCulloch-Pitts neurônios para calcular as funções lógicas E e OU.

A **Tabela A.1** ilustra o neurônio calculando **x** $\wedge$ **y**. De uma forma similar, a somatória das entradas multiplicadas aos respectivos pesos é feita pelo modelo que representa a função OU, veja **Tabela A.2** que retorna um valor maior ou igual a zero ao menos que **x** e **y** sejam iguais a 0.

x	y	x + y - 2	Saída
1	1	0	1
1	0	-1	-1
0	1	-1	-1
0	0	-2	-1

Tabela A.1 O modelo de McCulloch-Pitts para a função lógica E

x	y	x + y - 1	Saída
1	1	1	1
1	0	0	1
0	1	0	1
0	0	-1	-1

Tabela A.2 O modelo de McCulloch-Pitts para a função lógica OU

McCulloch e Pitts demonstraram o poder da computação neural, mas os interesses por esses estudos apenas começaram a ter sucesso com o desenvolvimento de algoritmos práticos de aprendizagem.

Os modelos mais antigos tiveram uma grande influência nos trabalhos do psicologista D. O. Hebb (1949), que especulou que o aprendizado no cérebro ocorreria através de modificações de sinapses. A proposta de Hebb é que se dois neurônios, por exemplo **A** e **B,** distam o suficiente de forma que **A** crie

estímulos constante ou persistentemente em **B** a ponto de ativá-lo, uma reação se desencadeia em uma ou ambas as células fortalecendo a eficiência de **A** como uma das células que provocou a ativação da célula **B**.

Psicologistas modernos tentaram implementar o modelo de Hebb porém falharam para produzir resultados gerais sem a adição de um mecanismo inibitório (Rochester et al. 1988, Quinlan 1991).

Apêndice B – Problema de Classificação em Redes Neurais Simples

Um exemplo de problema não linearmente separável é a função lógica ou-exclusivo ou XOR como mostra a **Tabela B.1**:

x_1	x_2	Saída
1	1	0
1	0	1
0	1	1
0	0	0

Tabela B.1 A tabela para a função lógica OU-exclusivo ou XOR

Considerando um perceptron com duas entradas x_1 e x_2, dois pesos w_1 e w_2, e um valor limite **t**, para que uma rede aprenda essa função, ela precisa encontrar valores para os pesos que satisfaçam as seguintes inequações de acordo com a **Tabela B.1**:

$$w_1*1 + w_2*1 < t$$
$$w_1*1 + 0 > t$$
$$0 + w_2*1 > t$$
$$0 + 0 < t$$

A série de inequações acima não tem solução. Isso prova que um perceptron não é capaz de resolver um problema envolvendo uma função OU-exclusivo ou XOR.

O que faz da função OU-exclusivo uma função impossível de ser resolvida pelo modelo perceptron de uma camada é que as duas classes a serem identificadas não são *linearmente separáveis*. Como mostra a **Figura** B.1,

é impossível traçar uma linha reta no plano bidimensional que separe os pontos ativos (pretos) dos pontos inibidos (brancos), isto é (1,0) e (0,1), e (0,0) e (1,1).

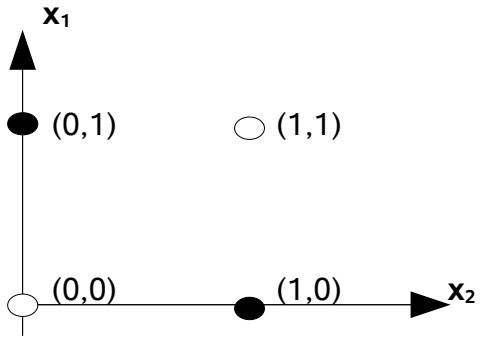

Figura B.1 Problema OU-exclusivo ou XOR

Também é possível pensar em um conjunto de valores de uma rede como uma definição de espaço. Cada parâmetro de uma entrada de dado corresponde a uma dimensão, onde cada valor de entrada define um ponto no espaço. No exemplo do OU-exclusivo, os quatro valores de entrada, representados pelas coordenadas x_1 e x_2, permitem montar a **Figura B.1.** A aprendizagem em uma classificação binária resume-se a separação dos valores de saída em dois grupos. Para um espaço de n dimensões, uma classificação é linearmente separável se suas classes puderem ser separadas por um hiperplano n – 1 dimensional. Em duas dimensões, um hiperplano n-dimensional é uma linha, em três dimensões é um plano, etc.

Inicialmente o modelo perceptron entusiasmou muitos, porém suas limitações foram analisadas por Nilsson (1965) e outros que identificaram que o modelo não era capaz de resolver problemas em certas classes de dificuldade, especificamente problemas em onde os pontos dos dados não são linearmente separáveis.

Essas limitações quanto a separação linear fez com que as pesquisas se

direcionassem novamente para as arquiteturas baseadas em símbolos, atrofiando de certa forma as pesquisas envolvendo a metodologia conexionista. Porém, trabalhos subsequentes entre 1970 e 1980 mostraram que esse problema tinha uma solução.

Apêndice C – Código

A linguagem de programação utilizada no estudo foi definida de forma a ter a melhor compatibilidade entre as aplicações. Dada a importância da biblioteca FANN (Nissen, S. 2009) para o estudo e suas raízes em C, essa foi a linguagem adotada para os códigos.

O código fonte tem três arquivos que podem ser compilados em C:

- funcoes_gerais.c;
- traffic_training.c;
- traffic_ga.c.

funcoes_gerais.c

```
////////////////////////////////////////////////////////////////////////
//
//          Esse arquivo contem funcoes utilizadas por outros algoritimos
//          vizando a reutilizacao das mesmas
//
//
#include <fann.h>
#include <stdio.h>
#include <sys/stat.h>
#include "floatfann.h"



// Definicoes de estruturas de dados

typedef struct errOrdering {
    int arrayIndex;
    float arrayError;
} EOR;

typedef struct errWeighArray {
    float neuronError;
    float neuronWeigh;
} EWARR;

typedef struct connWeight {
```

```
        int from_neuron, to_neuron, n_conn;
        float weight;
} CONNW;



// Funcoes gerais
// Le arquivo de erro
float GetError(char net[30])
{
   FILE *erron;
   float error;

   erron = fopen(net,"r");
   fscanf(erron,"%f\n",&error);
   fclose(erron);

   return(error);
}
FANN_EXTERNAL EWARR FannGetNeuronError(struct fann *ann, char *net)
{
   fann_type tmp_error;
   unsigned int i,j,k;
   struct fann_layer *layer_it;
   struct fann_neuron *neuron_it, *last_neuron;
   struct fann_neuron **connections;

   fann_type *error_begin = ann->train_errors;
   fann_type *error_prev_layer;
   fann_type *weights;
   const struct fann_neuron *first_neuron = ann->first_layer->first_neuron;
   const struct fann_layer *second_layer = ann->first_layer + 1;
   struct fann_layer *last_layer = ann->last_layer;
   EWARR **arrayErrors = (struct errWeighArray **) malloc(((int)ann->first_layer + 2) * sizeof(EWARR *));
      FILE *neuronError;


   /* go through all the layers, from last to first.
    * And propagate the error backwards */
      neuronError = fopen(net,"w");
      k = 0;
   for(layer_it = last_layer -1; layer_it >= second_layer; --layer_it)
   {
      last_neuron = layer_it->last_neuron;

      /* for each connection in this layer, propagate the error backwards */
      if(ann->connection_rate >= 1)
      {
```

```
    if(ann->network_type == FANN_NETTYPE_LAYER)
    {
        error_prev_layer = error_begin + ((layer_it - 1)->first_neuron - first_neuron);
    }
    else
    {
        error_prev_layer = error_begin;
    }
                j = 0;
    for(neuron_it = layer_it->first_neuron; neuron_it != last_neuron; neuron_it++)
    {
        arrayErrors[(int)layer_it] = (struct errWeighArray *) malloc(((int)(neuron_it->last_con - neuron_it->first_con) + 1) * sizeof(EWARR));
        tmp_error = error_begin[neuron_it - first_neuron];
        weights = ann->weights + neuron_it->first_con;
        for(i = neuron_it->last_con - neuron_it->first_con; i--;)
        {
            arrayErrors[(int)layer_it][i].neuronError = error_prev_layer[i];
            arrayErrors[(int)layer_it][i].neuronWeigh = weights[i];
                            fprintf(neuronError,"%d %d %d %f %f\n",k,j,i,error_prev_layer[i],weights[i]);
                    //printf("i = %d\n neuron_it = %d\n", i,neuron_it);
              //printf("error_prev_layer[%d] = %f\n", i, error_prev_layer[i]);
              //printf("weights[%d] = %f\n", i, weights[i]);
              error_prev_layer[i] += tmp_error * weights[i];
        }
                    j++;
    }
                k++;
}
 else
{
    for(neuron_it = layer_it->first_neuron; neuron_it != last_neuron; neuron_it++)
    {

        tmp_error = error_begin[neuron_it - first_neuron];
        weights = ann->weights + neuron_it->first_con;
        connections = ann->connections + neuron_it->first_con;
        for(i = neuron_it->last_con - neuron_it->first_con; i--;)
        {
            error_begin[connections[i] - first_neuron] += tmp_error * weights[i];
        }
    }
}
/* then calculate the actual errors in the previous layer */
 error_prev_layer = error_begin + ((layer_it - 1)->first_neuron - first_neuron);
 last_neuron = (layer_it - 1)->last_neuron;

 for(neuron_it = (layer_it - 1)->first_neuron; neuron_it != last_neuron; neuron_it++)
 {
```

```
            *error_prev_layer *= fann_activation_derived(neuron_it->activation_function,
                neuron_it->activation_steepness, neuron_it->value, neuron_it->sum);
            error_prev_layer++;
        }

    }
//return arrayErrors;
fclose(neuronError);
for(layer_it = last_layer -1; layer_it >= second_layer; --layer_it){
    free(arrayErrors[(int)layer_it]);
}
free(arrayErrors);
}



EOR *OrderErros(){

        const int size = ReadNetQant();
        EOR *errorOrdering;
    int j,i,arrayIndexTemp;
        float arrayErrorTemp;
    char err[30];

        if((errorOrdering = (struct errOrdering *) malloc((size + 1) * sizeof(EOR) )) != NULL){
        for (j=1;j<=size;j++){
                sprintf(err,"../nets/%d/traffic%d.err",j,j);
                        errorOrdering[j].arrayError = GetError(err);
                        errorOrdering[j].arrayIndex = j;
        // TP    printf("Err == %f   %d\n",errorOrdering[j].arrayError,errorOrdering[j].arrayIndex);
        }
                for (j=1;j<=size;j++) {
                        for (i=1;i<=size;i++){
                                if (errorOrdering[j].arrayError > errorOrdering[i].arrayError){
                                        arrayErrorTemp = errorOrdering[i].arrayError;
                                        arrayIndexTemp = errorOrdering[i].arrayIndex;
                                        errorOrdering[i].arrayError = errorOrdering[j].arrayError;
                                        errorOrdering[i].arrayIndex = errorOrdering[j].arrayIndex;
                                        errorOrdering[j].arrayError = arrayErrorTemp;
                        errorOrdering[j].arrayIndex = arrayIndexTemp;
                                }
                        }
                }
                return errorOrdering;
                free(errorOrdering);
        }

}
```

```
CONNW *GetConnectionWeight(int netID) {
        struct fann *ann;
        struct fann_connection *connections;
        char net[30], err[30];
        int j, num_connections;
        CONNW * connWeight;

        sprintf(net,"../nets/%d/traffic%d.net",netID,netID);
    sprintf(err,"../nets/%d/traffic%d.err",netID,netID);

    ann = fann_create_from_file(net);
    num_connections = fann_get_total_connections(ann);
    connections = (struct fann_connection *) malloc(sizeof(struct fann_connection) * num_connections);
        connWeight = (struct connWeight *) malloc(sizeof(struct connWeight) * (num_connections +1));
    if (connections != NULL)
            if(connWeight != NULL){
            fann_get_connection_array(ann,connections);
                    connWeight[0].n_conn=num_connections;

            for(j=0;j<=num_connections;j++){
                    connWeight[j].from_neuron=connections[j].from_neuron;
                    connWeight[j].to_neuron=connections[j].to_neuron;
                    connWeight[j].weight=connections[j].weight;
            }
        }
    fann_destroy(ann);
        return connWeight;
        free(connections);
        free(connWeight);
}

// Remove diretorios
int  RemoveDirectory(char *dir_name)
{

    remove(dir_name);

    return 0;
}

// Cria diretorios para novas redes
int CreateDirectory(char *dir_name)
{
    mode_t dir_mode = S_IRWXU | S_IRWXG | S_IRWXO;
    int ret = mkdir(dir_name, dir_mode);

    return 0;
```

```
}

Interact(int netID1,int netID2){

    FILE *fnet1,*fnet2,*erron;
  struct fann *ann;
  struct fann_connection *connection;
    char net1[30],net2[30],netB1[30],netB2[30],netB3[30],neuronerr[30],err[30];
  int j1,k1,i1,j2,k2,i2,i,j,num_connections,out,neuro;
    float error,error1,weight1,error2,weight2,newW;
    const       float desired_error = (const float) 0.001;
  const     unsigned int max_epochs = 10000;
  const     unsigned int epochs_between_reports = 1000;

    CONNW * connWeight;


  sprintf(net1,"../nets/%d/traffic%d.neu",netID1,netID1);
  sprintf(net2,"../nets/%d/traffic%d.neu",netID2,netID2);
  sprintf(netB1,"../nets/%d/traffic%d.net",netID1,netID1);
  sprintf(netB2,"../nets/%d%d",netID1,netID2);
  sprintf(netB3,"../nets/%d%d/traffic%d%d.net",netID1,netID2,netID1,netID2);

    connWeight = GetConnectionWeight(netID1);
    ann = fann_create_from_file(netB1);
    num_connections = fann_get_total_connections(ann);
  connection = (struct fann_connection *) malloc(sizeof(struct fann_connection) * num_connections + 1);

    fnet1 = fopen(net1,"r");
    fnet2 = fopen(net2,"r");

  while (fscanf(fnet1, "%d %d %d %f %f\n",&k1,&j1,&i1,&error1,&weight1) != EOF && fscanf(fnet2, "%d %d %d %f %f\n",&k2,&j2,&i2,&error2,&weight2) != EOF) {
        //printf("%d %d %d %f %f -- %d %d %d %f %f\n",k1,j1,i1,error1,weight1,k2,j2,i2,error2,weight2);
        if(k1==k2 && j1==j2 && i1==i2){
            if(error1 < 0){
                error1 = (error1 * (-1));
            }
            if(error2 < 0){
        error2 = (error2 * (-1));
      }
            if(error1 == error2){
                newW = (weight1 + weight2)/2;
            } else {
                if(error1 == 0 || (-1)*error1 == 0){
                    newW = weight1;
                } else {
                    if(error2 == 0 || (-1)*error2 == 0){
                newW = weight2;
```

```
                        } else {
                            if(error1 > error2){
                                newW = weight2;
                            } else {
                                newW = weight1;
                            }
                        }
                    }
            }
            }
            // Inicio dos cruzamentos
            //
            i = 0;
            while (connWeight[i].weight != weight1 && connWeight[i].weight) {
                i++;
            }
            if(connWeight[i].weight){
                connWeight[i].weight = newW;
            }
            //printf("Peso Novo:%0.5f ---- weight1= %0.5f weight2=%0.5f ---- Erros Erro1= %f Erro2= %f\n",newW,weight1,weight2,error1,error2);
    }
    for(j=0;j<num_connections;j++){
    connection[j].from_neuron=connWeight[j].from_neuron;
    connection[j].to_neuron=connWeight[j].to_neuron;
    connection[j].weight=connWeight[j].weight;
}
fann_set_weight_array(ann,connection,num_connections);
RemoveDirectory(netB2);
CreateDirectory(netB2);
fann_save(ann,netB3);


    fann_set_learning_rate(ann,0.2);
fann_train_on_file(ann, "../tdata/traffic.data", max_epochs, epochs_between_reports, desired_error);
// saida dos erros pra arquivo: "../%d%d/traffic%d%d.neu"
sprintf(NEURONERR,"../nets/%d%d/traffic%d%d.neu",netID1,netID2,netID1,netID2);
FannGetNeuronError(ann,neuronerr);
// saida do erro pra arquivo: "../%d%d/traffic%d%d.err"
error = fann_get_MSE(ann);
neuro = fann_get_total_neurons(ann);
out = fann_get_total_connections(ann);
sprintf(err,"../nets/%d%d/traffic%d%d.err",netID1,netID2,netID1,netID2);
erron = fopen(err,"w");
fprintf(erron,"%0.5f\n",error);

fann_destroy(ann);

fclose(erron);
```

```
        fclose(fnet1);
        fclose(fnet2);
        free(connection);

}

int ReadNetQant(){
    const char* index;
    struct stat info;
    int oldTotal = 0;
    FILE *numeron;

    index="../nets/indexRedes.txt";

    if (stat(index, &info) == 0)
    {
        numeron = fopen(index,"r+");
        fscanf(numeron,"%d",&oldTotal);
    }
    return(oldTotal);
    fclose(numeron);
}

RegisterNetQuantity(int newTotal){
    const char* index;
    struct stat info;
    int oldTotal = 0;
    FILE *numeron;

    index="../nets/indexRedes.txt";

    if (stat(index, &info) == 0)
    {
        numeron = fopen(index,"r+");
        fscanf(numeron,"%d",&oldTotal);
    } else {
        numeron = fopen(index,"w+");
    }
    newTotal = oldTotal + newTotal;
    fseek(numeron,0,SEEK_SET);
    fprintf(numeron,"%d\n",newTotal);
    fclose(numeron);
}
```

traffic_training.c

```
#include <fann.h>
#include <stdio.h>
#include <sys/stat.h>
#include "floatfann.h"


int main(int argc, char** argv)
{
    const        unsigned int num_input = 15;
    const        unsigned int num_hidden0 = 5;
    const        unsigned int num_hidden1 = 5;
    const        unsigned int num_output = 5;
    const        unsigned int num_layers = 4;
    const        float desired_error = (const float) 0.001;
    const        unsigned int max_epochs = 10000;
    const        unsigned int epochs_between_reports = 1000;
                int i,j,neuro;
                int out;
                float error;
                char net[50],err[50],neuronerr[50];
                char fold[50];
                FILE *erron;
                void write_each_neuron_err();

                typedef struct errWeighArray {
                float neuronError;
                float neuronWeigh;
                } EWARR;

                EWARR FannGetNeuronError();


                if(argc >= 2){
                    i = ReadNetQant() + 1;
                    j = atoi(argv[1]) + i;
                    for (i;i<j;i++)
                    {
                    struct fann *ann = fann_create_standard(num_layers, num_input, num_hidden0, num_hidden1, num_output);
                    fann_set_activation_function_hidden(ann, FANN_SIGMOID);
                    fann_set_activation_function_output(ann, FANN_SIGMOID);
                        fann_set_learning_rate(ann,0.3);
                    fann_train_on_file(ann, "../tdata/traffic.data", max_epochs, epochs_between_reports, desired_error);
                        sprintf(fold,"../nets/%d",i);
                        RemoveDirectory(fold);
                        CreateDirectory(fold);
                        sprintf(neuronerr,"../nets/%d/traffic%d.neu",i,i);
                        sprintf(net,"../nets/%d/traffic%d.net",i,i);
                      FannGetNeuronError(ann,neuronerr);
```

```
                        // saida do erro pra arquivo: "../%d/traffic%d.err"
                        error = fann_get_MSE(ann);
                        neuro = fann_get_total_neurons(ann);
                        out = fann_get_total_connections(ann);
                        sprintf(err,"../nets/%d/traffic%d.err",i,i);
                        erron = fopen(err,"w");
                        fprintf(erron,"%0.5f\n",error);
                        fclose(erron);
                        //fim da saida, o erro final sera gravado
                        fann_save(ann, net);

                        fann_destroy(ann);
                    }
                RegisterNetQuantity(atoi(argv[1]));
            } else
                printf("####\n\n####\n\n##\nDefina o numero de redes neurais a serem geradas \nEx. ./traffic_training 10 para gerar 10 redes\n\n####\n\n##\n#");

            return 0;
}
```

traffic_ga.c

```
#include <fann.h>
#include <stdio.h>
#include "floatfann.h"
#include <sys/stat.h>

typedef struct errOrdering {
        int arrayIndex;
        float arrayError;
} EOR;

typedef struct connWeight {
        int from_neuron, to_neuron, n_conn;
        float weight;
} CONNW;


int main(int argc, char** argv)
{
    EOR *orderedArray, *orderedUnique, *OrderErros();
    CONNW *connectionWeight,*GetConnectionWeight();
        fann_type *calc_out;
        fann_type input[15];
        char net[30], err[30];
        int i, j, k, num_connections;
```

```
    float   GetError();
    struct fann_connection *connections;
    //struct fann *ann;
    const int size = ReadNetQant();



    // Organiza as redes em ordem decrescente, ordenada por erros
  orderedArray = OrderErros();
    if((orderedUnique=(struct errOrdering *) malloc((size + 1) * sizeof(EOR) )) == NULL)
        return 0;


    // Define grupos de erros onde cada grupo contem uma amostra de cada incidencia
    k=1;
    i=0;
  while (i < size){
        i++;
        j=i+1;
        while (j<=size){
            if (orderedArray[i].arrayError != orderedArray[j].arrayError) {
                orderedUnique[k].arrayError = orderedArray[i].arrayError;
                orderedUnique[k].arrayIndex = orderedArray[i].arrayIndex;
                k++;
                i=j-1;
                j=size;
            }
            j++;
        }
  }
    orderedUnique[k].arrayError = orderedArray[i].arrayError;
  orderedUnique[k].arrayIndex = orderedArray[i].arrayIndex;
    for(i=1;i<=k;i++){
            printf("%f  %d\n",orderedUnique[i].arrayError,orderedUnique[i].arrayIndex);
    }
    // Executa cruzamento genetico
    if(size > 1){
        for (i=1;i <= size;i++)
            for (j=1;j<=size;j++)
                if(i != j){
                    Interact(i,j);
                }
        {
    }
    return 0;
    } else {
        printf("Redes devem ser criadas antes de executar o algoritmo GA");
    }
}
```

Além do código existem *scripts* que foram utilizados para facilitar a execução dos processos bem como para gerar as curvas:

- gcc_ga_traffic.sh – gera arquivos binarios para traffic_training.c e traffic_ga.c;
- plot.sh – é utilizado pelo script ploter.sh;
- ploter.sh – gera curvas de erros e pesos para as Redes.

O conteúdo dos arquivos acima listados é:

gcc_ga_traffic.sh

```
#!/bin/bash
##
#
#
#
gcc -O3 -lm -l fann traffic_training.c funcoes_gerais.c -o traffic_training
gcc -O3 -lm -l fann traffic_ga.c funcoes_gerais.c -o traffic_ga
```

plot.sh

```
#!/bin/sh
###
#
#
###
gnuplot -persist<< EOF
set output "$1.png";
set terminal x11;
set title "Net $1" textcolor lt 3
plot "$1" using 1:2 title "Erros" w l, "$1" using 1:3 title "Pesos" w l;
EOF
```

ploter.sh

```
#!/bin/bash
##
#
#
#
########
cd ../nets
LIST=`ls -l| grep drwxr | cut -d" " -f8`
cd ../actual
for i in $LIST
```

```
do
        cd ../nets/$i
        awk '{print $4,$5}' traffic$i.neu| grep -n . | awk -F: '{print $1,$2}' > traffic$i.neu.plot
        cd ../../actual
        ./plot.sh ../nets/$i/traffic$i.neu.plot
done
```

Apêndice D – Considerações na Implementação e Extração de Dados

Quanto a extração dos dados de um ambiente real, deve ser considerado que a cada sinal de trânsito tenha ao menos um contador terrestre de veículos pertencente a ele. O posicionamento ideal para o contador de cada objeto está entre o sinal e o(s) ponto(s) gerador(es) ou extrator(es) de tráfego adjacente(s) a esse. O contador deverá ser situado nas proximidades do ponto adjacente.

O motivo de se manter a maior distância entre os pontos é captar o maior volume de tráfego possível, evitando assim uma falha no processamento devido a entradas incoerentes. Nessa análise, uma precisão do volume de tráfego é de extrema importância, pois esse fator será de grande influência nas tomadas de decisões das redes.

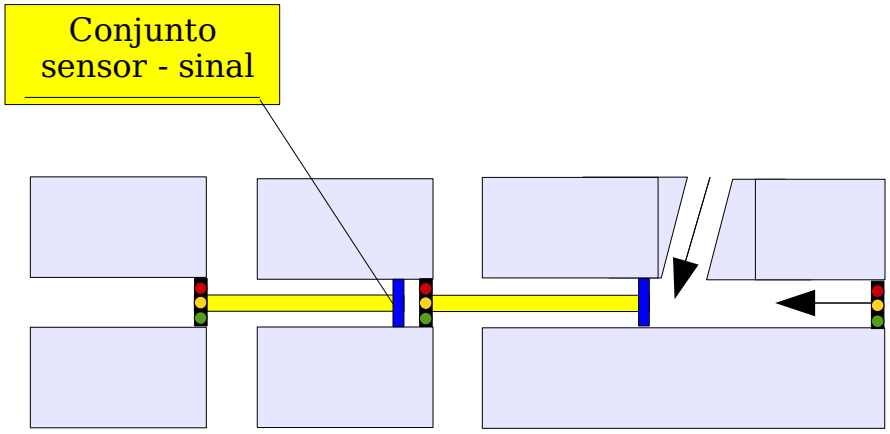

Figura D.1 Posicionamento dos sensores

Apêndice E – Gráficos de Erros x Pesos

Cruzamentos entre as Redes Neurais:

- primeira Rede Neural com as demais;

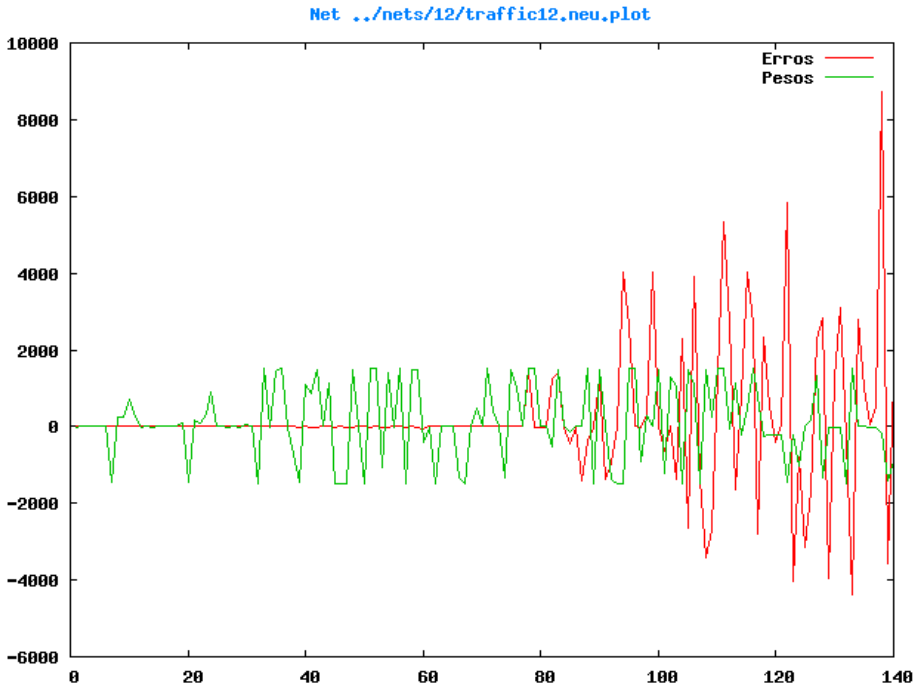

Figura E.1 Gráfico comparando pesos e erros do cruzamento 1x2

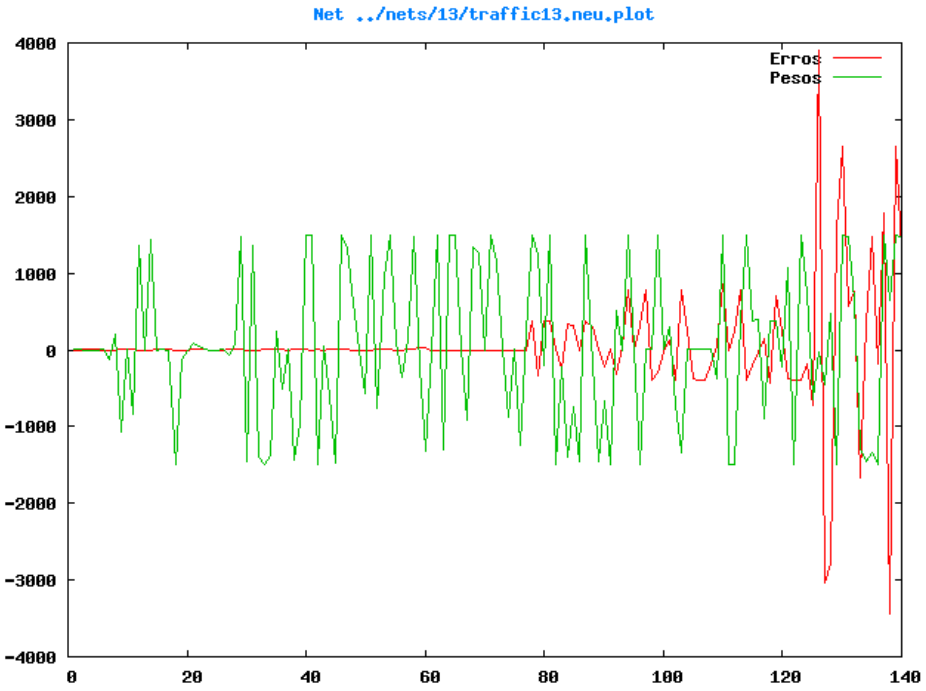

Figura E.2 Gráfico comparando pesos e erros do cruzamento 1x3

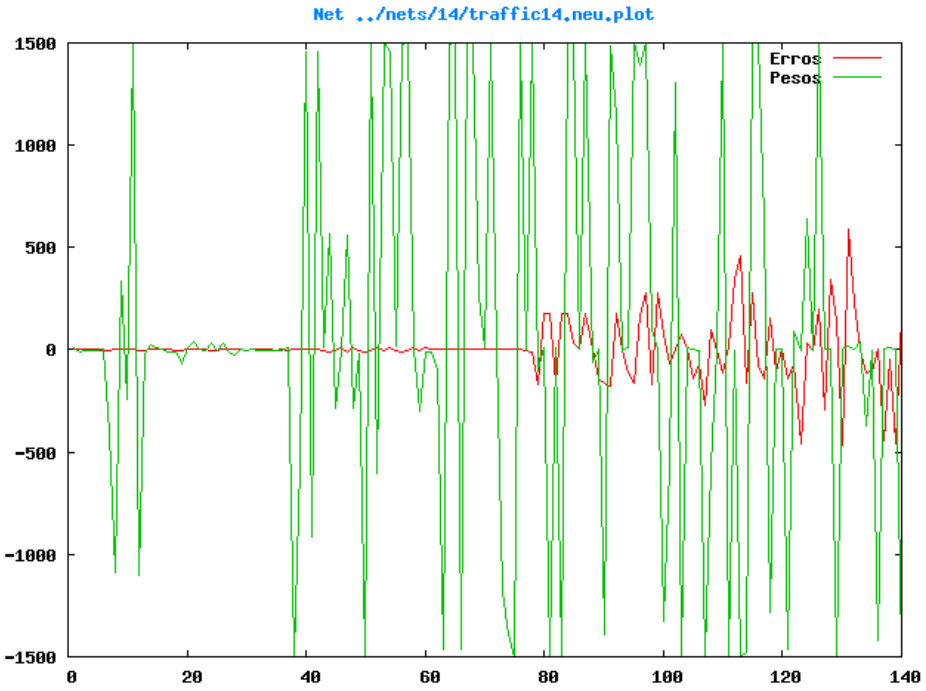

Figura E.3 Gráfico comparando pesos e erros do cruzamento 1x4

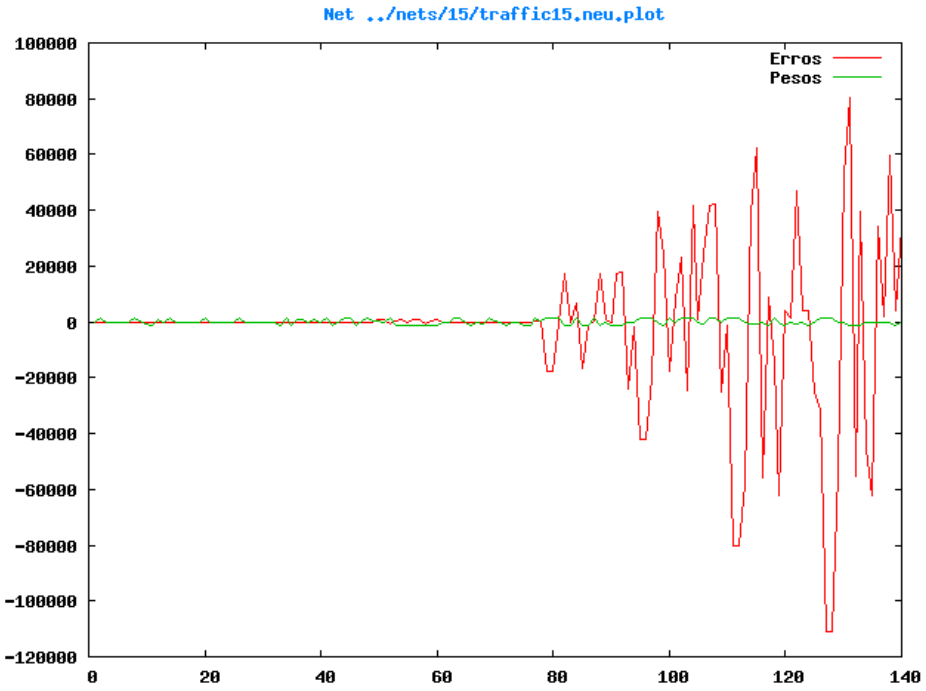

Figura E.4 Gráfico comparando pesos e erros do cruzamento 1x5

- segunda Rede Neural com as demais;

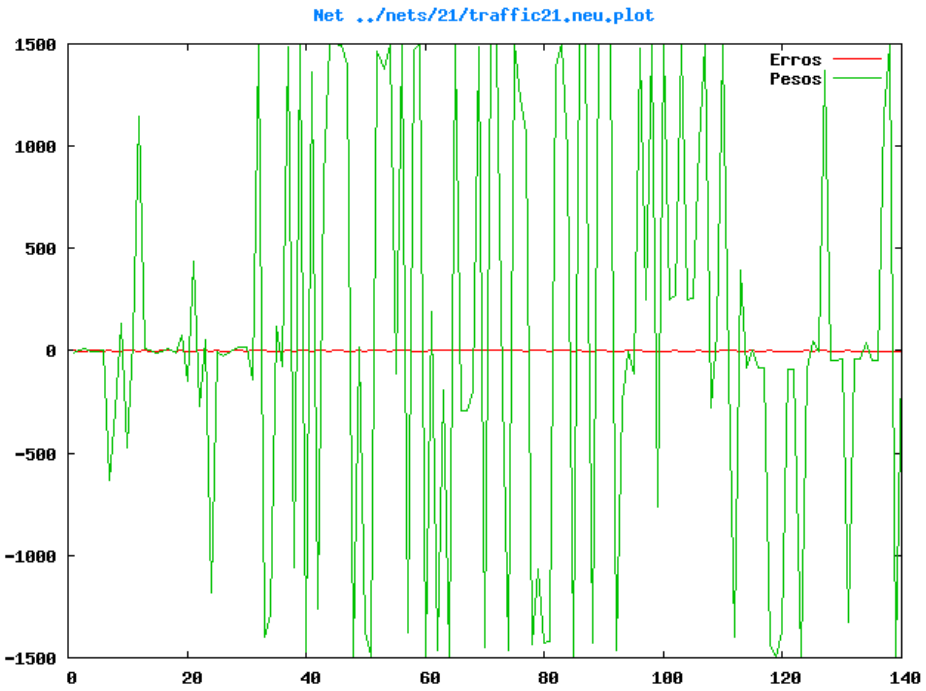

Figura E.5 Gráfico comparando pesos e erros do cruzamento 2x1

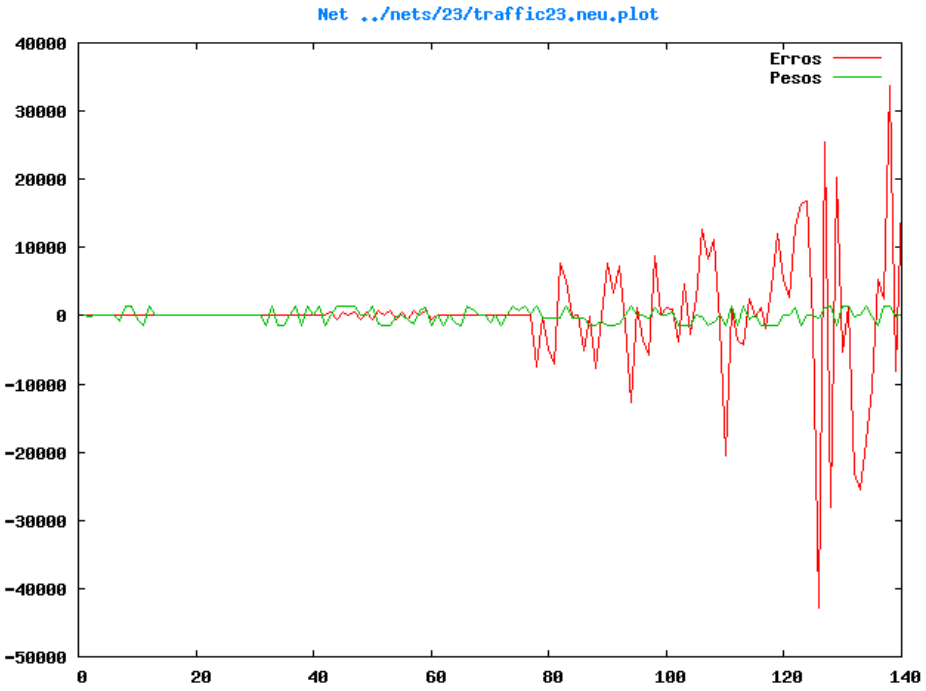

Figura E.6 Gráfico comparando pesos e erros do cruzamento 2x3

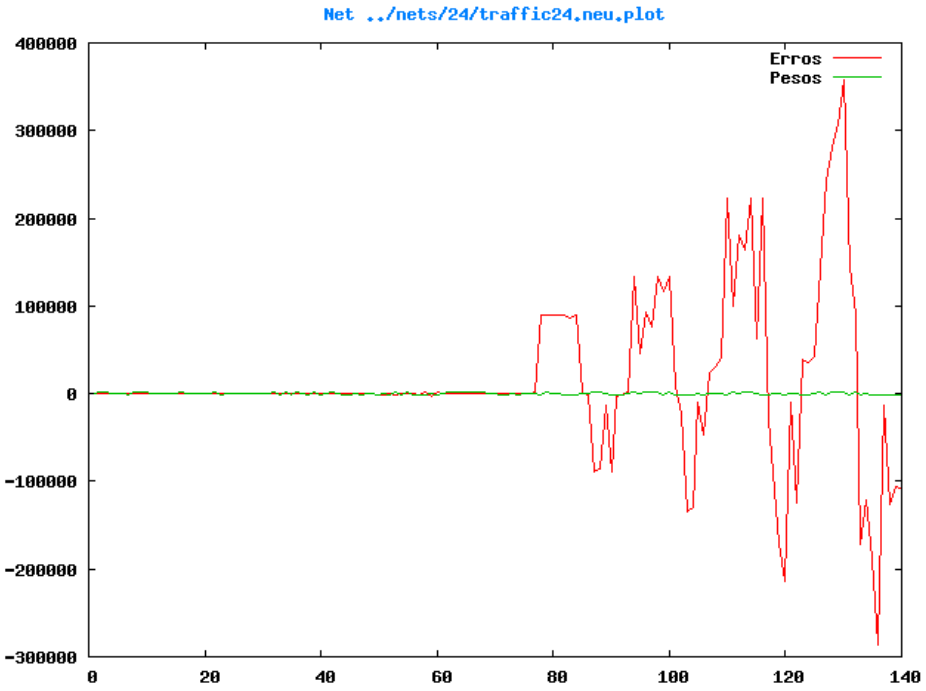

Figura E.7 Gráfico comparando pesos e erros do cruzamento 2x4

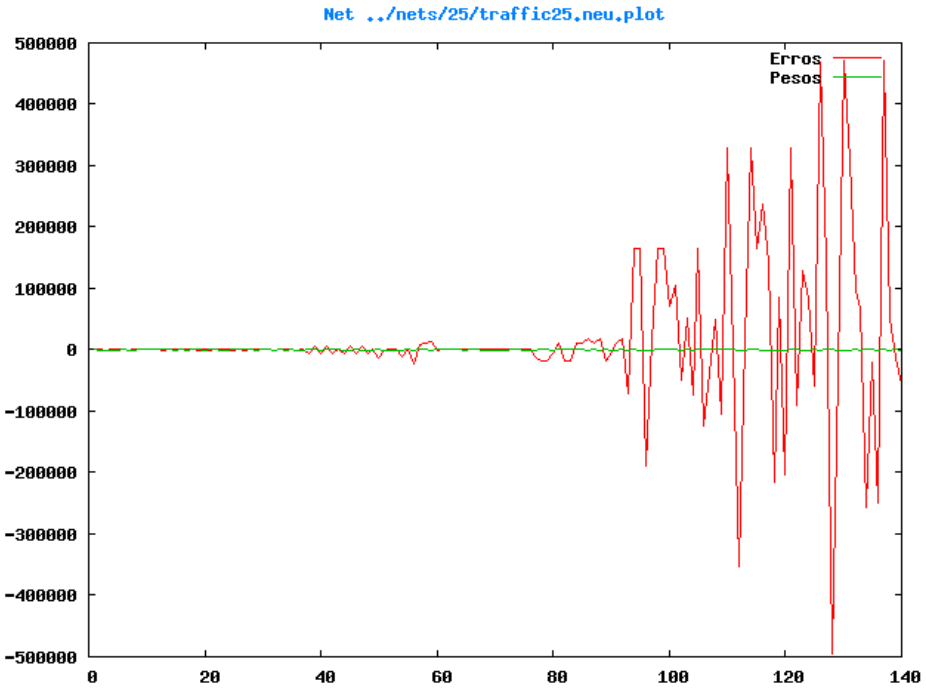

Figura E.8 Gráfico comparando pesos e erros do cruzamento 2x5

- terceira Rede Neural com as demais;

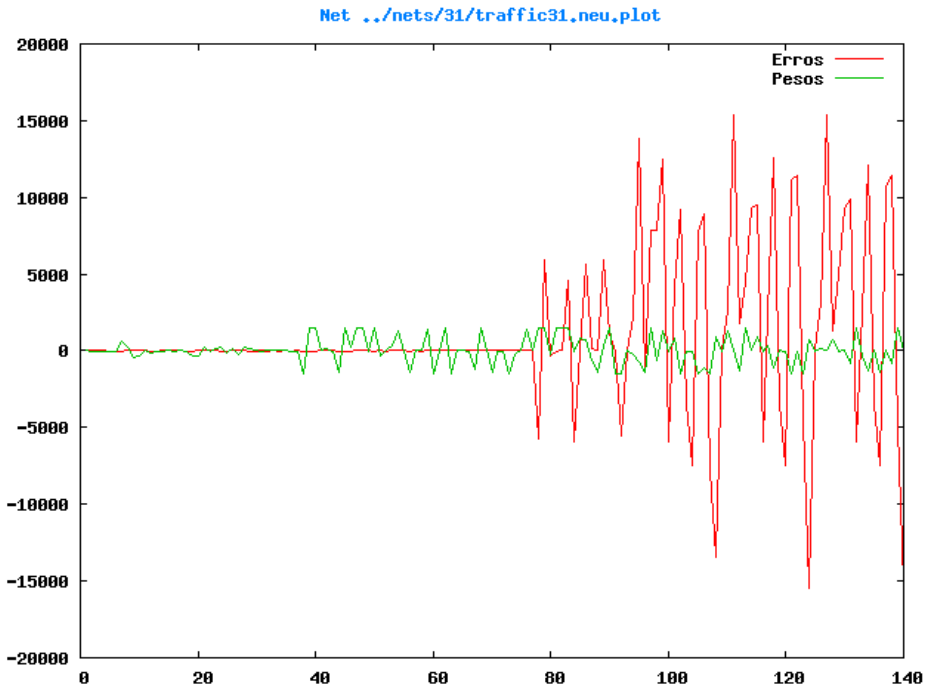

Figura E.9 Gráfico comparando pesos e erros do cruzamento 3x1

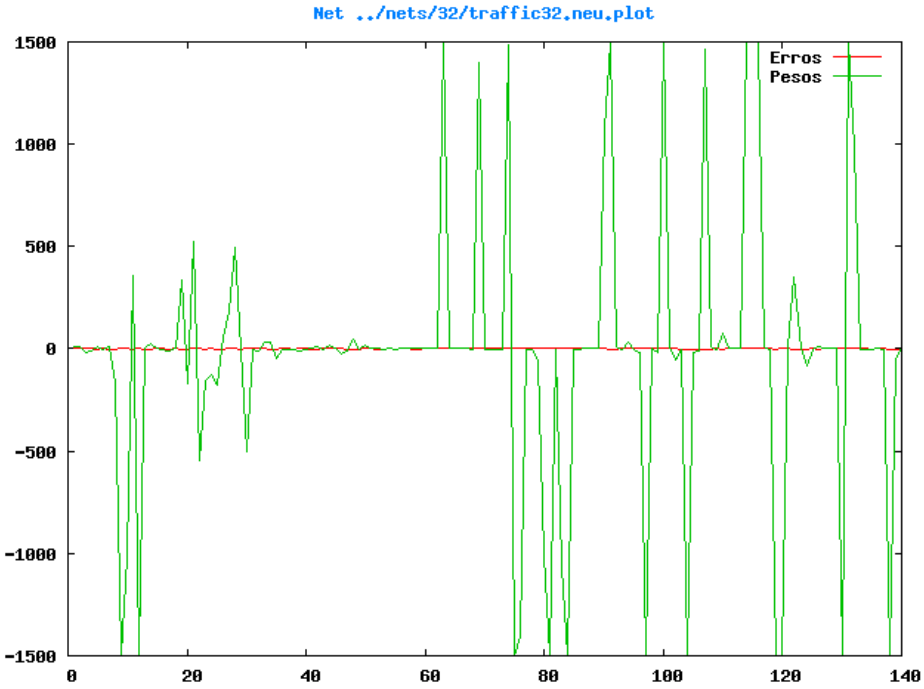

Figura E.10 Gráfico comparando pesos e erros do cruzamento 3x2

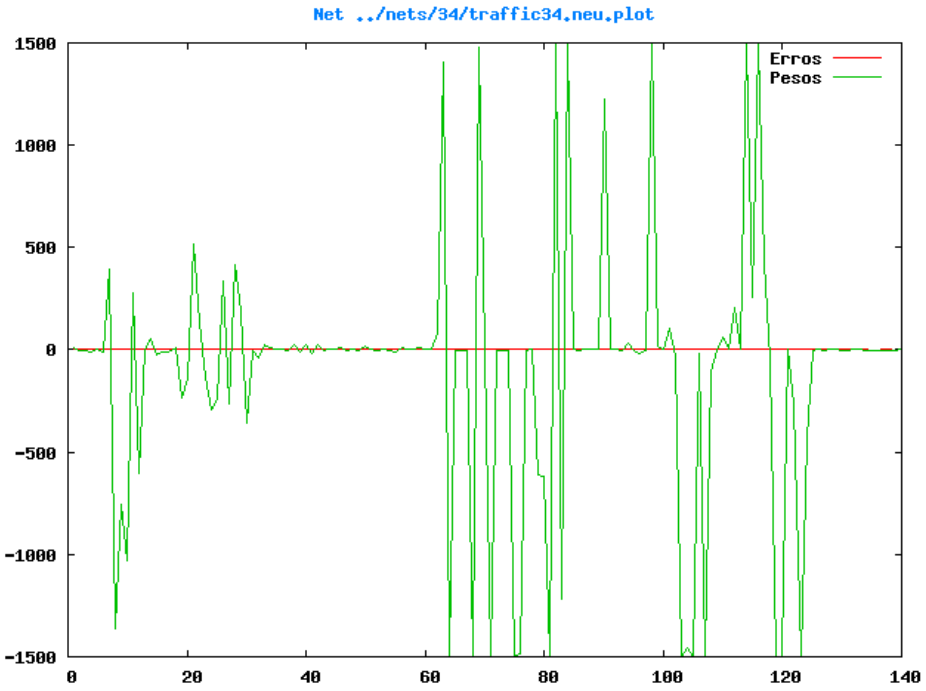

Figura E.11 Gráfico comparando pesos e erros do cruzamento 3x4

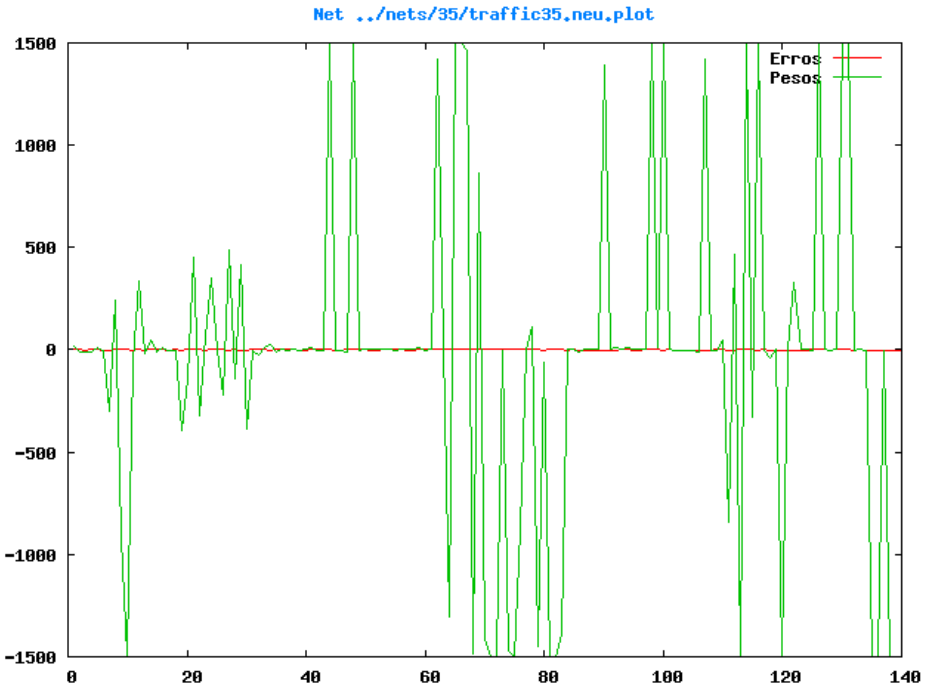

Figura E.12 Gráfico comparando pesos e erros do cruzamento 3x5

- quarta Rede Neural com as demais;

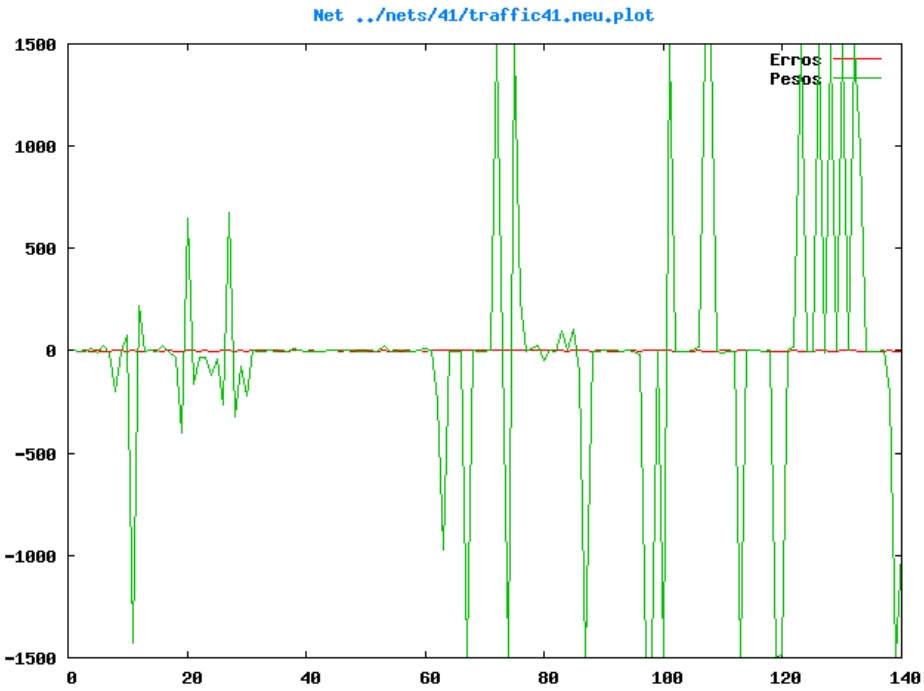

Figura E.13 Gráfico comparando pesos e erros do cruzamento 4x1

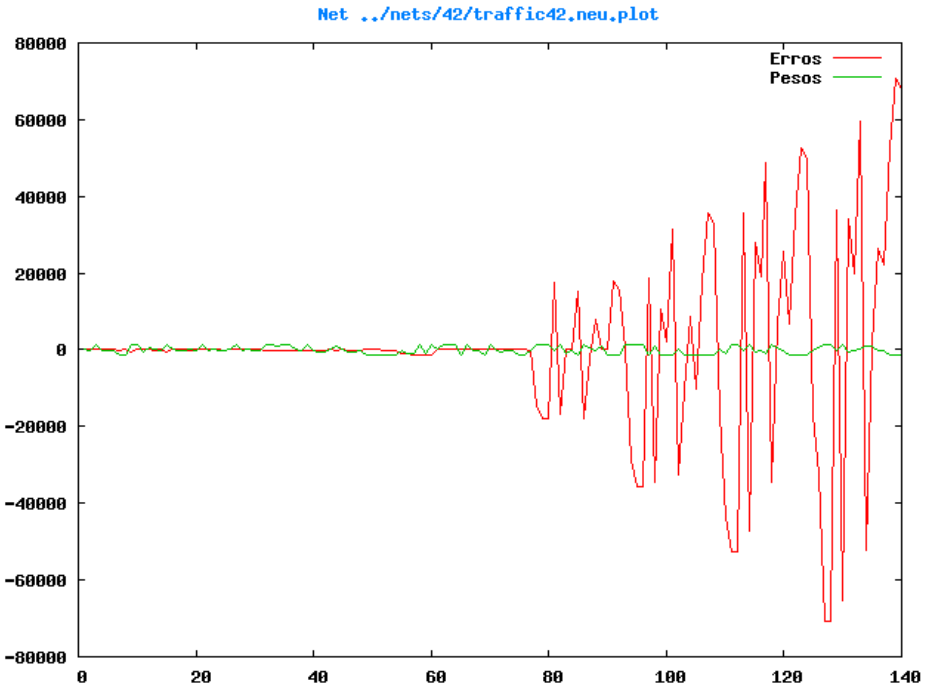

Figura E.14 Gráfico comparando pesos e erros do cruzamento 4x2

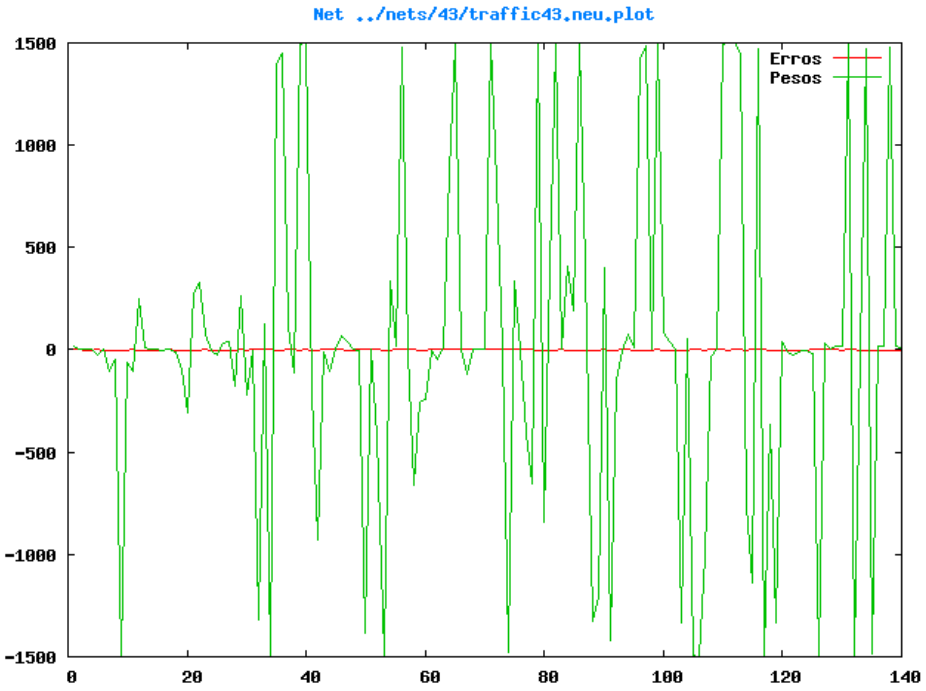

Figura E.15 Gráfico comparando pesos e erros do cruzamento 4x3

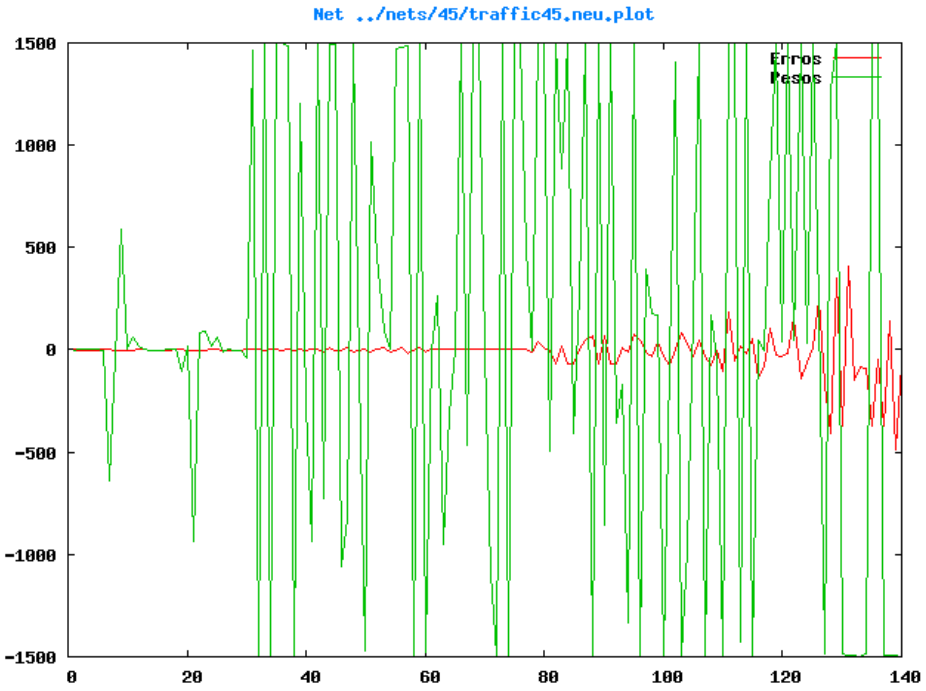

Figura E.16 Gráfico comparando pesos e erros do cruzamento 4x5

- quinta Rede Neural com as demais;

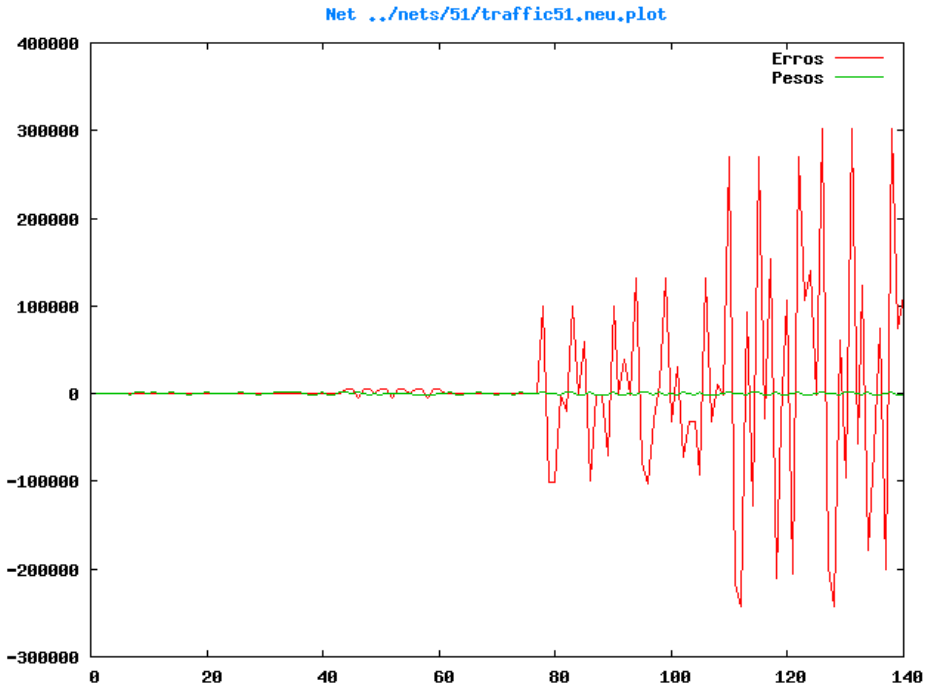

Figura E.17 Gráfico comparando pesos e erros do cruzamento 5x1

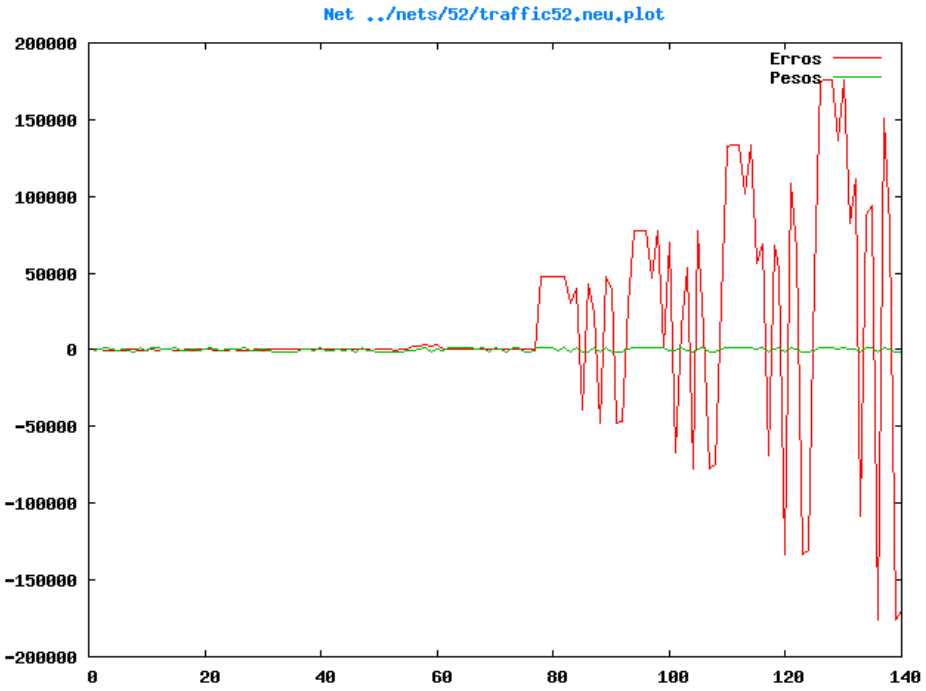

Figura E.18 Gráfico comparando pesos e erros do cruzamento 5x2

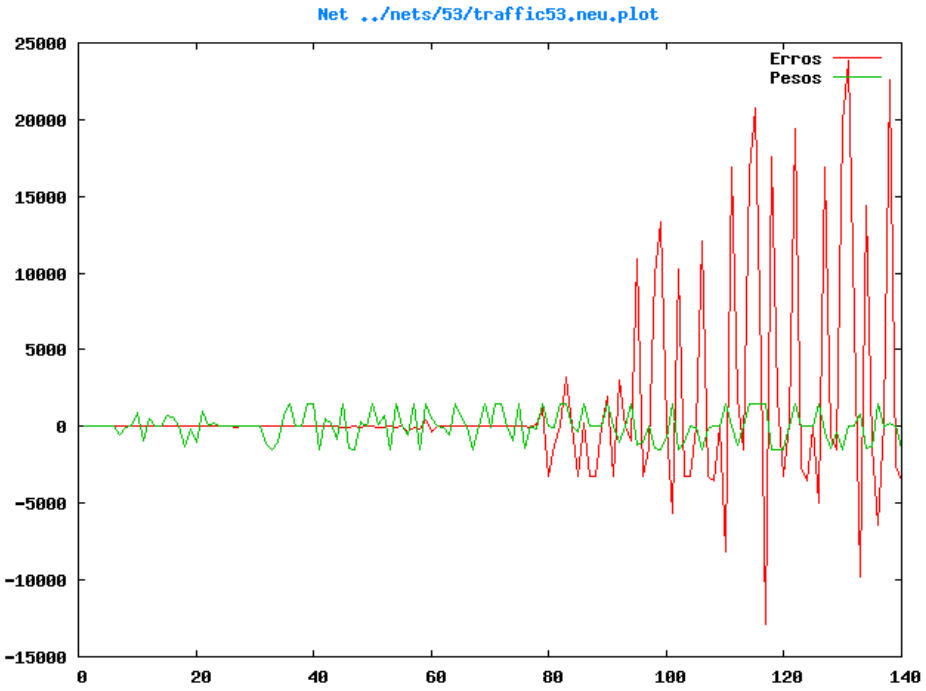

Figura E.19 Gráfico comparando pesos e erros do cruzamento 5x3

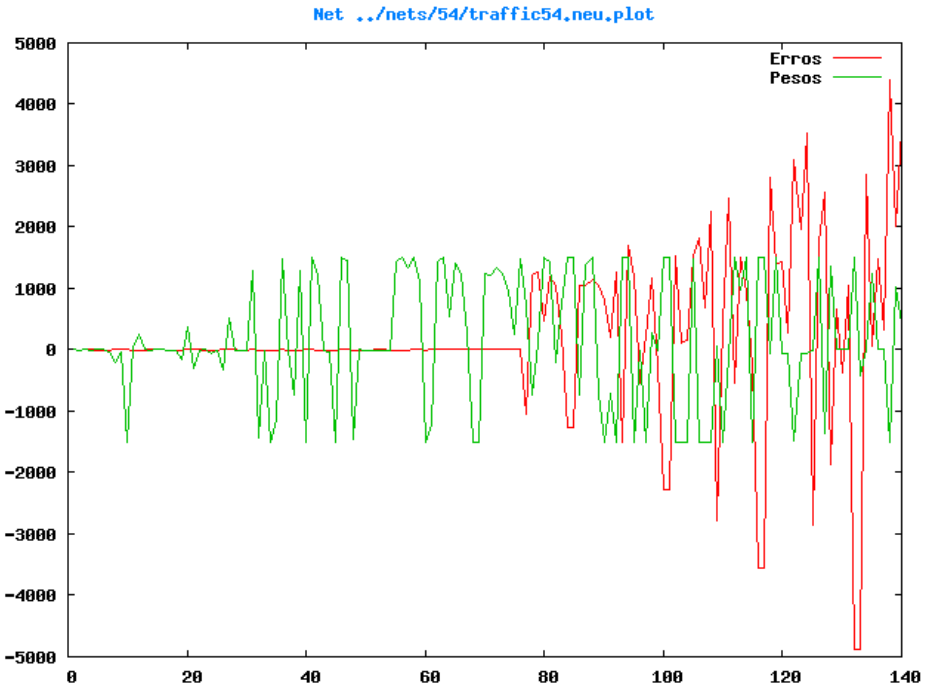

Figura E.20 Gráfico comparando pesos e erros do cruzamento 5x4