UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
CURSO DE CIÊNCIA DA COMPUTAÇÃO
DEPARTAMENTO DE ELETRÔNICA E COMPUTAÇÃO

APLICAÇÃO DE EVOLUÇÃO ESTRUTURAL E PARAMÉTRICA DE REDES NEURAIS DINÂMICAS NA MELHORIA DE DESEMPENHO DE MÉTODOS DE ASSIMILAÇÃO DE DADOS

TRABALHO FINAL DE GRADUAÇÃO

André Grahl Pereira

Santa Maria
2011

APLICAÇÃO DE EVOLUÇÃO ESTRUTURAL E PARAMÉTRICA DE REDES NEURAIS DINÂMICAS NA MELHORIA DE DESEMPENHO DE MÉTODOS DE ASSIMILAÇÃO DE DADOS

por

André Grahl Pereira

Trabalho de Final de Graduação apresentado ao Curso de Ciência da Computação, da Universidade Federal de Santa Maria (USFM, RS), como requisito parcial para a obtenção de grau de
Bacharel em Ciência da Computação.

Orientador: Profª. Drª. Juliana Kaizer Vizzotto
Co-Orientador: Dr. Adriano Petry

Trabalho de Graduação 321
Santa Maria, RS, Brasil
2011

Universidade Federal de Santa Maria
Centro de Tecnologia
Curso de Ciência da Computação
Universidade Federal de Santa Maria

A Comissão Examinadora, abaixo assinada,
aprova o Trabalho Final de Graduação

APLICAÇÃO DE EVOLUÇÃO ESTRUTURAL E PARAMÉTRICA DE REDES NEURAIS DINÂMICAS NA MELHORIA DE DESEMPENHO DE MÉTODOS DE ASSIMILAÇÃO DE DADOS

elaborado por

André Grahl Pereira

como requisito parcial para obtenção do grau de
Bacharel em Ciência da Computação

Comissão Examinadora:

_____
Prof[a]. Dr[a]. Juliana Kaizer Vizzotto (UFSM)
(Presidenta/Orientadora)

_____
Dr. Adriano Petry (INPE)
(Co-Orientador)

_____
Prof. Dr. Marcos Cordeiro d'Ornellas (UFSM)

_____
Prof[a]. Dr[a]. Andrea Schwertner Charão (UFSM)

Santa Maria 2011

“Disciplina, Determinação e Dedicação.”
— ANDRÉ GRAHL PEREIRA

AGRADECIMENTOS

Agradeço primeiramente a mim. Depois a Seiya de Pégaso por mostrar que se suficientemente determinado pode-se vencer até mesmo um deus. Por fim a Bruce Wayne por mostrar que com disciplina e dedicação pode-se superar até mesmo um alienígena virtualmente invencível.

RESUMO

Trabalho de Graduação
Curso de Ciência da Computação
Universidade Federal de Santa Maria

APLICAÇÃO DE EVOLUÇÃO ESTRUTURAL E PARAMÉTRICA DE REDES NEURAIS DINÂMICAS NA MELHORIA DE DESEMPENHO DE MÉTODOS DE ASSIMILAÇÃO DE DADOS

Autor: André Grahl Pereira
Orientador: Profa. Dra. Juliana Kaizer Vizzotto
Co-Orientador: Dr. Adriano Petry

Data e Local da Defesa: 16 de dezembro de 2011, Santa Maria.

O uso de modelos de previsão numéricos são essenciais para a sociedade moderna. A assimilação de dados é uma técnica que tem por objetivo aumentar a eficiência de previsão destes modelos, combinando dados de modelo com dados oriundos de observações, obtendo um estado que mais se aproxima do real estado da natureza. Combinar estas duas fontes de informação, dados observacionais e de modelo, tem se mostrado um desafio, mesmo para supercomputadores presentes neste tipo de aplicação. Assim, redes neurais vêm sendo propostas como uma alternativa, objetivando uma assimilação de alta qualidade a um menor custo computacional. Esse trabalho propõe investigar o modelo neuroevolucionista NEAT em assimilação de dados. NEAT é capaz de adaptar, usando princípios de computação evolutiva, os pesos das conexões, bem como a topologia da rede neural buscando uma topologia mínima e obtendo melhor desempenho. Dessa forma, foi desenvolvido um software para testes que possibilitou a avaliação da abordagem proposta. Através do experimento no Atractor de Lorenz verificou-se que o modelo NEAT foi capaz de emular a tarefa de assimilação de dados com um menor erro quando comparado com a rede neural treinada por *backpropagation* e a partir do experimento com o modelo de Água Rasa observou-se que o modelo NEAT sempre obtém uma topologia com um numero significativo menor operações e com estados suficientemente grandes também opera a um menor custo computacional.

Palavras-chave: NEAT, redes neurais, assimilação de dados

ABSTRACT

Undergraduate Final Work
Graduation in Computer Science
Federal University of Santa Maria

APPLICATION OF PARAMETRIC AND STRUCTURAL EVOLUTION OF NEURAL NETWORK DYNAMICS IN IMPROVING PERFORMANCE OF METHODS OF DATA ASSIMILATION

Author: André Grahl Pereira
Advisor: Profª. Drª. Juliana Kaizer Vizzotto
Co- Advisor: Dr. Adriano Petry

Date and Local: December 16th 2011, Santa Maria.

The use of numerical prediction models are essential to modern society. Data assimilation is a technique that aims to increase the efficiency of prediction of these models by combining model data with data from observations, obtaining a state that is closer to the true state of nature. Combining these two sources of information, observational and model data, has been a challenge, even for supercomputers present in this type of application. Thus neural networks have been proposed as an alternative to perform the assimilation of high quality at a lower computational cost. This paper proposes to investigate the neuroevolucionist model NEAT in data assimilation. NEAT is able to adapt, using principles of evolutionary computation, the weights of the connections and the topology of the neural network in a search for a minimum topology and getting better performance. So, it was developed a software that enabled the test and evaluation of the proposed approach. Through the experiment on the Lorenz Attractor was found that the model NEAT was able to emulate the task of data assimilation with a smaller error when compared with the neural network trained by backpropagation from the experiment on the Shallow Water model was observed that NEAT model always gets a topology with significantly fewer operations and states with large enough also operates at a lower computational cost.

Keywords: NEAT, neural networks, data assimilation

LISTA DE FIGURAS

LISTA DE TABELAS

NENHUMA ENTRADA DE ÍNDICE DE ILUSTRAÇÕES FOI ENCONTRADA.

LISTA DE ABREVIATURAS E SIGLAS

LCCE - *Laboratório de Computação para o Clima Espacial*

GPS - *Global Positioning System*

TWEANNs - Topology *and Weight Evolving Artificial Neural Networks*

NEAT - NeuroEvolution of Augmenting Topologies

BLUE - *Best Linear Unbiased Estimator*

ÍNDICE

1 INTRODUÇÃO

Modelos de previsão numéricos são essenciais para a sociedade moderna. É através destes que se torna possível a previsão de importantes fenômenos físicos. No entanto existe uma grande imprevisibilidade associada a estes fenômenos. Nesse contexto, a assimilação de dados surge como uma metodologia que tem por objetivo aumentar a eficiência de previsão destes modelos (Nowosad, 2001). A previsão numérica em geral depende de uma representação precisa do estado inicial, e um pequeno erro neste estado pode causar grandes erros no estado previsto (Daley, 1991).

Assimilação de dados é uma metodologia que propõe uma maneira eficiente de combinar dados de um modelo de previsão numérico com dados oriundos de observações da natureza, obtendo um estado que mais se aproxima do real estado da natureza. Atualmente existe uma grande quantidade de observações disponíveis a estes modelos. Estas observações são oriundas de balões, barômetros, radares, GPS, ionosondas e satélites, dentre outras. As técnicas de assimilação de dados, além de combinar de forma eficiente esses dados, devem gerar o estado que mais se aproxima do real estado da natureza, minimizando erros originados em instrumentos observacionais ou erros de modelagem.

Modelos numéricos de previsão de interesse prático, como os modelos de clima terrestre e clima espacial, possuem um elevado custo computacional e suas previsões geram na ordem de 10^{12} dados (Cintra R. S., 2010). Combinar estas duas fontes massivas de informação, dados observacionais e de modelo, tem se mostrado um desafio, mesmo para supercomputadores presentes neste tipo de aplicação (Cintra, Velho, & Todling, 2009). Diante deste desafio, novas técnicas de assimilação de dados vêm sendo propostas, objetivando uma assimilação de alta qualidade a um menor custo computacional. Dessa forma, redes neurais têm sido propostas recentemente, apresentando resultados consistentes: são computacionalmente eficientes e eficazes quanto a sua aplicação (Cintra & Velho, 2010).

Redes neurais artificiais são modelos matemáticos ou computacionais inspirados em aspectos funcionais ou estruturais de redes neurais biológicas. Uma rede neural consiste de um grupo interconectado de neurônios processando informação através de uma abordagem conexionista. São empregadas em modelagem de padrões complexos entre entrada e saída ou para encontrar padrões em dados. Nessa abordagem, redes neurais são usadas como emuladores dos métodos clássicos de assimilação de dados. As redes neurais são treinadas a

partir de um conjunto de exemplos fornecido por um método clássico de assimilação de dados. Após o treinamento ser concluído, a rede neural é usada no processo de assimilação de dados, gerando um ganho de desempenho computacional.

Sendo essa uma abordagem recente, nem todas as suas possibilidades já foram exploradas. Em toda a bibliografia verificada apenas um trabalho (Härter, 2004) usou um modelo de rede neural que difere do modelo clássico: rede neural direta com uma camada oculta treinada através de *backpropagation*. No trabalho de (Härter, 2004) foram empregadas redes neurais recorrentes, o que implicou em um significativo ganho de desempenho na fase de treinamento e uma redução no número de neurônios necessários a tarefa. Esta abordagem aumentou o desempenho na fase de uso de fato da rede neural, mas mesmo nesse trabalho os modelos usados apresentavam topologia fixa. Nenhum dos trabalhos estudados explorou o conceito de topologia mínima, objetivando um ganho máximo de desempenho computacional. Assim, esse trabalho propõe investigar o modelo neuroevolucionista NEAT (Stanley & Miikkulainen, 2002) em assimilação de dados. O NEAT é capaz de adaptar, usando princípios de computação evolutiva, os pesos das conexões, bem como a topologia da rede neural.

Para os testes da abordagem proposta foi desenvolvido de um software de teste, contendo os seguintes módulos: modelos físicos, redes neurais e métodos de assimilação de dados. Os modelos físicos usados na comparação de desempenho computacional foram os modelos de Lorenz e de Água Rasa. Estes modelos foram escolhidos por serem referências da literatura em testes de novas técnicas de assimilação de dados, o modelo de Lorenz por apresentar um comportamento caótico e sensível às condições iniciais já o modelo de Água Rasa é adequado, pois o tamanho de estado pode ser aumentado arbitrariamente possibilitando testes de desempenho computacional. Ambos os modelos serão comparados em relação à eficácia e eficiência das seguintes técnicas de assimilação de dados: (1) B*est linear unbiased estimator (BLUE)*, rede neural direta treinada por *backpropagation*, (2) rede neural direta treinada por algoritmos genéticos e NEAT.

1.1 Objetivos

Este trabalho tem como objetivo geral investigar a técnica NEAT como solução do problema de assimilação de dados em função de sua eficácia e eficiência. Para realizar esta investigação foi desenvolvido um software que permite a análise comparativa com outras metodologias de assimilação de dados em dois modelos físicos.

1.1.1 Objetivos Específicos

- Estudo e implementação de diferentes modelos de redes neurais, sendo o foco o modelo neuroevolucionista NEAT;
- Implementação do modelo de Lorenz, que é referência em testes de eficácia de novas técnicas de assimilação de dados, e do modelo de Água Rasa, referência em testes de eficiência computacional;
- Implementação do método de assimilação de dados: *Best linear unbiased estimator*;
- Realização dos testes de eficiência e eficácia;
- Avaliação dos resultados obtidos a partir do aplicativo desenvolvido;

1.2 Organização do Texto

O Capítulo 2 apresentará a fundamentação teórica sobre os assuntos abordados no trabalho, sendo estes redes neurais, modelos numéricos e assimilação de dados, o Capítulo 3 irá discorrer sobre a justificativa e relevância do trabalho, bem como descreverá os principais componentes e módulos do software desenvolvido. Por fim, no Capítulo 4 serão apresentados os resultados para os testes efetuados e no Capítulo 5 as conclusões finais do estudo e previsões acerca de trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo serão descritos os conceitos teóricos que fundamentam o presente trabalho, assim como as técnicas utilizadas para o seu desenvolvimento. Será dado um foco especial para o modelo NEAT, que é a base do trabalho.

2.1 Redes Neurais Artificiais

Esta seção descreve os principais modelos de redes neurais, partindo da formulação inicial e apresentando os conceitos fundamentais de seu funcionamento.

2.1.1 Neurônio Artificial e Redes Neurais Artificiais

O estudo do cérebro humano levou ao desenvolvimento de modelos matemáticos que possuem o objetivo de compreender e reproduzir artificialmente o seu funcionamento. O modelo fundamental é o neurônio (Figura 2.1), sendo esta a unidade central de processamento de uma rede neural, cuja ativação é dada por:

$$\boldsymbol{u} = \boldsymbol{\sigma}\left(\sum_{\boldsymbol{i=1}}^{\boldsymbol{n}} \boldsymbol{w_i x_i}\right)$$

onde $\boldsymbol{w_i}$ representa o ganho sináptico e $\boldsymbol{x_i}$ indica o valor das $\boldsymbol{n}$ entradas, que podem ter origem externa (sensores) ou interna (uma conexão recorrente) (Haykin, 2001). A função $\boldsymbol{\sigma}$ é denominada função de ativação ou limiar, a saída $\boldsymbol{u}$ de um neurônio assume o valor 1, se o campo local induzido daquele neurônio é não-negativo, e caso contrário 0, como mostrado a seguir:

$$\boldsymbol{\sigma} = \begin{cases} \boldsymbol{1\ se\ \sigma(x,w) \geq 0} \\ \boldsymbol{0\ se\ \sigma(x,w) < 0} \end{cases}$$

Tal modelo de neurônio é referido na literatura como o modelo de *McCulloch-Pitts* (Haykin, 2001). Apesar de sua simplicidade, esse modelo é capaz de ser treinado em um número finito de iterações para classificar um conjunto de dados linearmente separáveis.

A partir desse modelo inicial, vários outros foram propostos. Na década de 1980, o trabalho de Rumelhart e McClelland (Haykin, 2001) tornou famosa umas das técnicas mais conhecidas para o treinamento de neurônios de múltiplas camadas, o algoritmo de retropropagação de erro (ou *backpropagation*), tornando possível classificar um conjunto de dados não linearmente separáveis. Outros trabalhos também são famosos, como o de Hopfield

(1982), onde é aplicado o princípio de armazenamento de informação em redes dinamicamente estáveis (memória associativa), e o trabalho de Kohonen (1982), que apresenta uma arquitetura capaz de reduzir a dimensão de um espaço $\boldsymbol{n}$-dimencional preservando sua topologia (mapa de Kohonen).

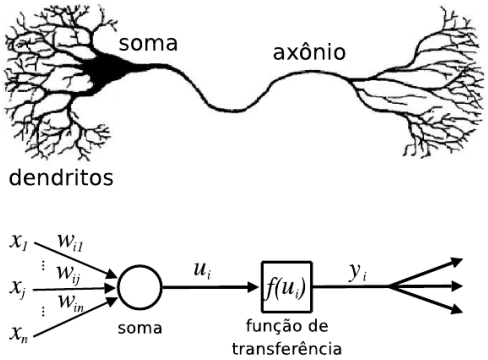

Figura 2.1: Neurônio biológico e seu modelo matemático (Miguel, 2009).

2.1.2 Topologias de Redes Neurais

A topologia da rede neural define como os neurônios se conectam para formar uma rede de neurônios, existindo dois principais tipos de topologia, as redes diretas (*feedforward*) e as redes neurais recorrentes (*feedback*).

Redes neurais diretas (Figura 2.2) são aquelas que possuem um grafo de conectividade associado que não possui ciclos. Essas redes são comumente representadas em formato de camadas, com a informação se propagando em uma única direção. Os neurônios são excitados camada a camada. Neste caso a rede se comporta de forma estática, mapeando um espaço $\mathbb{R}^m$ em um espaço $\mathbb{R}^n$.

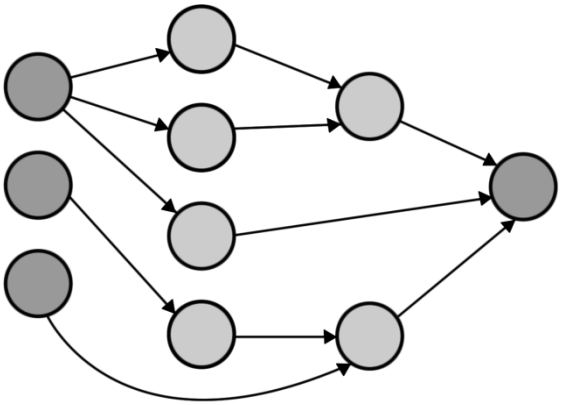

Figura 2.2: Exemplo de rede neural direta.

Já as redes neurais recorrentes (Figura 2.3) ou realimentadas são definidas como aquelas que possuem um grafo de conectividade que contem um ou mais ciclos. Apresentam um comportamento dinâmico, mesmo sem nenhuma excitação e a partir de entradas a rede neural pode continuar a produzir saídas. Dessa forma as conexões recorrentes mapeiam algo semelhante a uma memória.

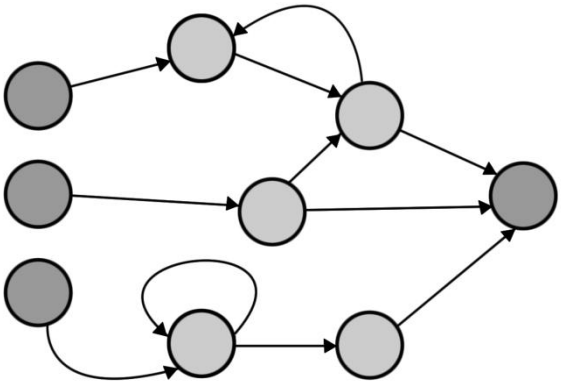

Figura 2.3: Exemplo de rede neural recorrente.

2.1.3 Aprendizado em Redes Neurais

A aprendizagem é um processo fundamental para redes neurais e é através deste que a rede neural pode melhorar seu desempenho. Uma rede neural aprende sobre seu ambiente através de um processo interativo em que ajustes sucessivos são aplicados às conexões sinápticas. O tipo de aprendizagem é definido pela forma como os ajustes dos pesos são realizados. Assim, os processos de aprendizagem são classificados em três principais categorias: supervisionado, não supervisionado e por reforço.

A aprendizagem supervisionada, também chamada de aprendizagem com professor, é um processo em que é fornecido um conjunto de exemplos entrada-saída. Assim os pesos sinápticos são ajustados de forma a reproduzir a tabela de treinamento. Um exemplo deste tipo de aprendizagem é o algoritmo de retropropagação do erro (Russell & Norvig, 2010).

Já na aprendizagem não supervisionada ou sem professor, a rede neural recebe as entradas e determina uma saída de acordo com uma regra de aprendizagem. Não existe um ponto definido para qual cada entrada deve corresponder, por isso a regra deve ser capaz de extrair as características do conjunto de entradas. Esse tipo de aprendizagem é frequentemente empregado em mineração de dados (Rojas, 1996). Um exemplo desse modelo é dado pelo princípio de Hebb (Haykin, 2001).

Por fim, na aprendizagem por reforço o mapeamento de entrada-saída é realizado através da interação com ambiente. O sistema é projetado para aprender por reforço atrasado, ou seja, a rede neural é executada em um determinado ambiente por certo período de tempo, sendo avaliada por um "crítico" ao fim de cada iteração. Os pesos da rede neural são então ajustados de acordo com avaliação desse "crítico". Esse método é frequentemente empregado em técnicas neuroevolucionistas como NEAT.

2.1.4 Ativação de Redes Neurais

A ativação de redes neurais diretas (*feedforward*) é realizada a partir da camada de entrada, de camada em camada até a saída, onde todos os neurônios de uma camada são ativados no mesmo instante. No entanto, em uma rede neural com uma topologia arbitrária a situação é mais complexa, já que existem ciclos. Assim um neurônio pode usar sua própria saída em sua entrada. Nesse caso é necessário modelar a rede neural como um sistema dinâmico discreto (Miguel, 2009), onde a cada instante de tempo a rede neural possui um estado diferente, podendo ser representado por um vetor onde cada componente é a saída de um neurônio. Para esse modelo existem duas formas de ativação: assíncrona e síncrona. Na forma assíncrona um neurônio arbitrário é escolhido e ativado com o atual estado da rede. Já na forma síncrona todos os neurônios são ativados usando o estado anterior da rede neural.

2.2 Evolução e Algoritmos Genéticos

Algoritmos genéticos (Buckland, 2002) consistem em uma técnica de busca, utilizada para encontrar soluções aproximadas em problemas de otimização. Esses pertencem a uma

classe particular de algoritmos evolutivos, que usam técnicas inspiradas pela biologia como hereditariedade, mutação, seleção natural e recombinação.

2.2.1 Principais Conceitos de Algoritmos Genéticos

Em algoritmos genéticos, cada indivíduo é descrito por um conjunto de valores denominado cromossomo. Os cromossomos (Figura 2.4) são compostos por genes, que armazenam as características que formarão o fenótipo. Cada gene pode assumir um dentre vários valores, chamados alelos. Uma configuração do cromossomo é denominada genótipo.

Algoritmos genéticos operam diretamente sobre o cromossomo abstraindo qualquer relação direta com fenótipo. Diante dessa flexibilidade muitos problemas podem ser modelados através dessa metodologia.

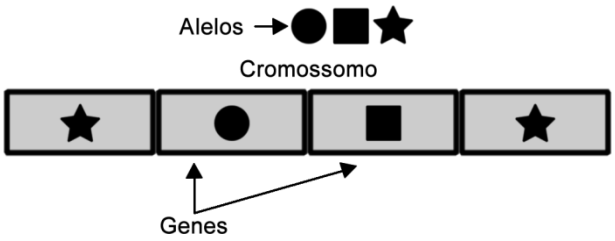

Figura 2.4: Representação de cromossomo, alelos e genes.

2.2.2 Modelo Básico de Algoritmo Genético

O modelo geral de algoritmo genético (Russell & Norvig, 2010) é padrão para a grande maioria de problemas em que é aplicado, e as diferenças de cada problema são restritas à representação e aos operadores genéticos aplicados.

A estrutura do algoritmo genético inicia com a geração aleatória de uma população inicial, esta população é submetida a um critério de avaliação. Se um determinado limiar de desempenho foi alcançado por um ou mais indivíduos, o algoritmo é encerrado. Caso contrário, ele continua o processo, sendo aplicados na sequência os operadores genéticos de seleção, reprodução, recombinação e mutação. Ao fim da iteração uma nova população é gerada e o processo continua até que uma condição de parada seja satisfeita, como ilustrado na Figura 2.5.

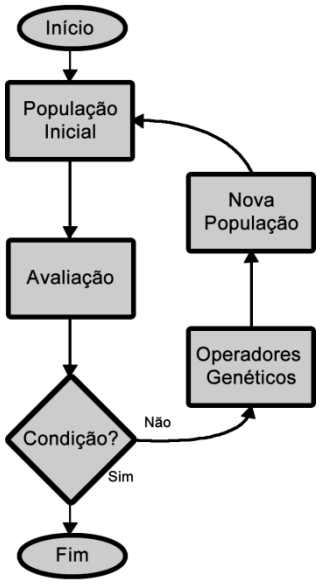

Figura 2.5: Modelo geral de algoritmo genético.

2.2.3 Seleção

Em algoritmos genéticos o método usado para seleção dos indivíduos que serão aplicados aos operadores genéticos é de fundamental importância. Três importantes métodos de seleção utilizados atualmente são: truncada, torneio e roleta (Buckland, 2002).

No método de seleção truncada os indivíduos da população são ordenados segundo seu desempenho, em seguida uma porção fixa dos indivíduos menos aptos são descartados. Na seleção por torneio um subconjunto do conjunto da população é selecionado aleatoriamente, ordenado sendo o individuo mais apto é selecionado. Por fim no método da roleta cada indivíduo tem uma probabilidade de ser selecionado segundo seu valor adaptativo.

2.2.4 Codificação

Um dos desafios na aplicação de algoritmos genéticos é a escolha da codificação adequada, já que esta deve representar de forma satisfatória o fenótipo a ser alcançado. Ademais, a escolha da codificação determina diretamente o conjunto de operadores genéticos que possivelmente serão utilizados. A codificação pode representar um caminho em um labirinto, um percurso em um grafo ou a topologia de uma rede neural. Cada problema exige uma codificação específica e assim um conjunto de operadores genéticos particulares.

2.2.5 Operadores Genéticos

Os operadores genéticos têm como objetivo imitar os processos naturais de reprodução e mutação, sendo assim responsáveis pela inserção de variabilidade genética na população (Buckland, 2002).

O operador de recombinação, ou *crossover*, imita o processo de reprodução. No modelo padrão de algoritmo genético são combinados dois indivíduos e a partir de troca de material genético são gerados dois novos indivíduos (Figura 2.6).

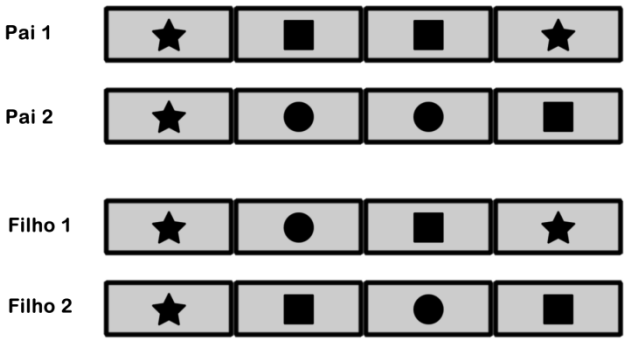

Figura 2.6: Descendentes recebendo metade da carga genética de cada pai.

Já o operador de mutação é responsável por inserir perturbações nos novos genótipos, aumentado assim a variabilidade genética da população. O modelo de operador de mutação pode variar muito em função da codificação empregada. No modelo padrão um gene é aleatoriamente trocado por outro possível (Figura 2.7).

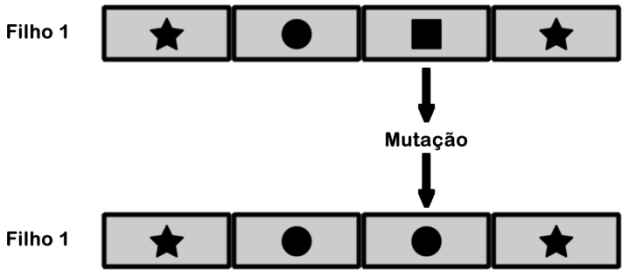

Figura 2.7: Representação para operador de mutação.

2.3 Neuroevolução

Neuroevolução é uma forma de evoluir uma rede neural empregando algoritmos genéticos, sendo que esses podem ser empregados na evolução dos pesos, da topologia ou de ambos.

A topologia da rede neural tem grande influência em seu funcionamento. Apesar de uma rede neural completamente conectada com uma camada oculta poder aproximar qualquer função contínua (Cybenko, 1989), trabalhos como o de Stanley (Stanley & Miikkulainen, 2002) mostram que a escolha da topologia adequada pode melhorar de forma significativa seu desempenho.

Diferentemente das técnicas de aprendizagem, não existe um metodologia eficiente e suficientemente genérica capaz de determinar a melhor forma com que os neurônios devam ser conectados (Miguel, 2009).

Na ultima década, vários métodos foram propostos para evolução da topologia e conexões. Esses métodos são genericamente conhecidos como *Topology and Weight Evolving Artificial Neural Networks* (TWEANN), e empregaram combinações de algoritmos genéticos e redes neurais com o objetivo de evoluir pesos e topologia. A seguir são descritos os principais desafios dessa abordagem, e em seguida discutidos os tipos de neuroevolução e por fim o método NEAT é apresentado.

2.3.1 Codificação Genética de Redes Neurais

Em um primeiro momento é necessário definir como o fenótipo (rede neural) será representado em forma de genótipo, já que o algoritmo genético opera apenas sobre o genótipo. Existem duas principais categorias de codificação, direta e indireta.

A codificação direta é a abordagem mais empregada pelos métodos TWEANNs. Nela os cromossomos armazenam toda a informação necessária para construção do fenótipo, ou seja, a rede neural o e genótipo representam a mesma estrutura armazenada de duas formas diferentes, uma em qual o algoritmo genético pode operar e outra na forma em que o indivíduo pode ser avaliado.

A forma mais simples de codificação direta de uma rede neural na forma de um cromossomo é através de uma lista de genes (Figura 2.8). Fixando a topologia da rede neural,

cada conexão é mapeada para uma posição do cromossomo. As desvantagens dessa representação são a necessidade de uma topologia fixa e o não conhecimento do algoritmo genético sobre a disposição dos neurônios.

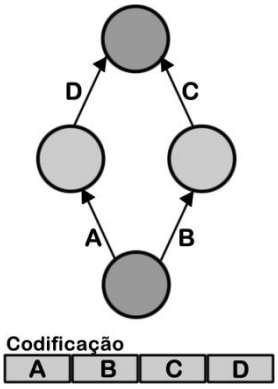

Figura 2.8: Representação para codificação direta de topologia fixa através de lista de genes.

Outra forma de codificação direta é a representação matricial, com uma matriz de adjacência onde o valor 1 na posição i, j representa que existe uma conexão entre o neurônio i e j. Essa representação apresenta como desvantagens o grande custo computacional, já que o tamanho da matriz é o quadrado do número de neurônios e no caso da adição de um neurônio a matriz precisa ser recriada.

Já a codificação indireta é aquela onde apenas as regras de construção são armazenadas no genótipo, assim o fenótipo pode ser construído a partir dessas regras. A codificação indireta permite uma representação mais compacta quando comparada com a codificação direta, já que cada conexão com cada neurônio não é especificada no genoma.

2.3.2 Problema da Permutação

O problema da permutação é um dos principais desafios na neuroevolução, ele ocorre porque duas redes neurais que resolvem o mesmo problema podem ter representações genicas diferentes. Dessa forma, genes podem ser perdidos durante a permutação, tornando os filhos, resultado da combinação, menos eficientes que os pais.

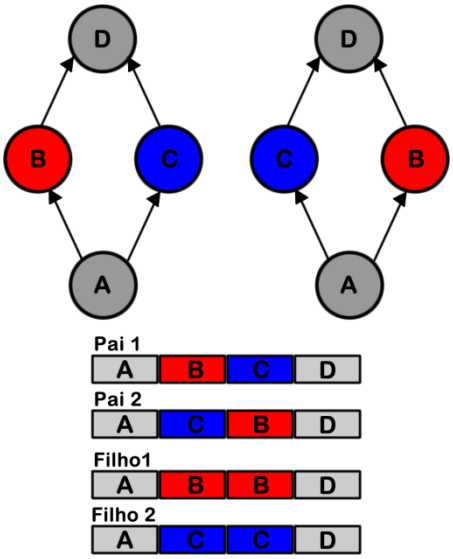

Figura 2.9: Exemplo de problema da permutação.

Na figura 2.9 pode-se ver um exemplo para o problema da permutação. Nela ambos os cromossomos possuem os mesmos neurônios e resolvem o mesmo problema, mas estes neurônios se encontram em posições diferentes na rede neural. Assim quando o fenótipo é mapeado para o genótipo, neurônios de mesma categoria funcional são mapeados para posições diferentes, gerando descendentes danificados através da permutação. O problema reside no mapeamento de neurônios que desempenham mesmas funções em uma rede neural.

2.4 NEAT

NEAT (*NeuroEvolution of Augmenting Topologies*) é o principal método utilizado neste trabalho e será discutido em detalhes no texto a seguir. Esse método evolui pesos e topologia de uma rede neural através de três premissas: aplicação de um operador de permutação eficiente, mesmo entre topologias diferentes; proteção de inovações estruturais utilizando especiação; e crescimento incremental a partir de uma estrutura mínima.

Dentre os métodos de neuroevolução, NEAT apresenta os melhores resultados em problemas clássicos como XOR e pendulo invertido (Stanley & Miikkulainen, 2002). O texto a seguir utilizou como referência o trabalho publicado por Stanley e Miikkulainen em 2002.

2.4.1 Codificação Genética

A codificação empregada em NEAT é a codificação direta, onde são utilizadas duas listas, uma armazenando os neurônios presentes na rede neural e outra armazenando todas as conexões. Essa codificação não impõe nenhuma restrição à topologia da rede neural, que pode apresentar estruturas e tamanho arbitrários.

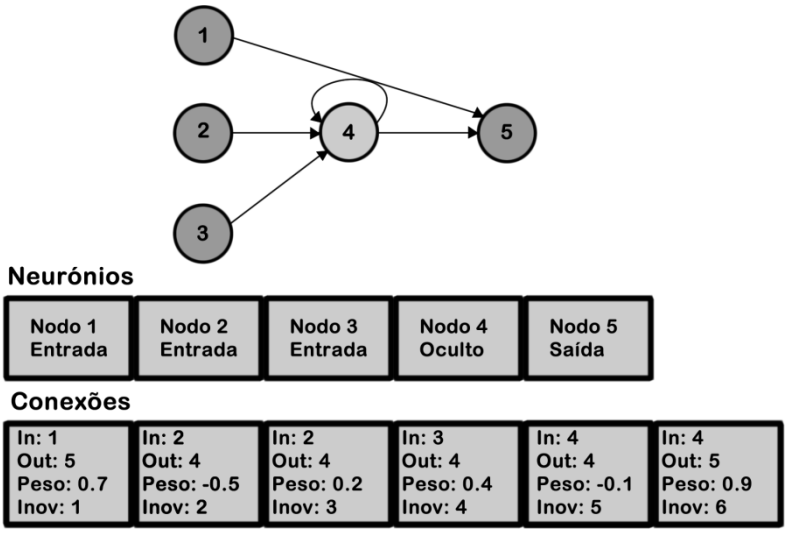

Figura 2.10: Exemplo de codificação empregada pelo método NEAT.

A figura 2.10 exemplifica a codificação empregada. Na lista de neurônios cada nodo possui um identificador único, bem como o tipo: entrada, oculto ou saída. Na lista de conexões são armazenadas de onde inicia a conexão, o destino dela e o peso sináptico. Outro dado armazenado na lista de conexões é sua identificação de inovação, esse número é atribuído a cada conexão que surge por operações de mutação e é através desta identificação de inovação que é possível realizar os cruzamentos de redes neurais de topologia diferentes.

2.4.2 Operador de Mutação

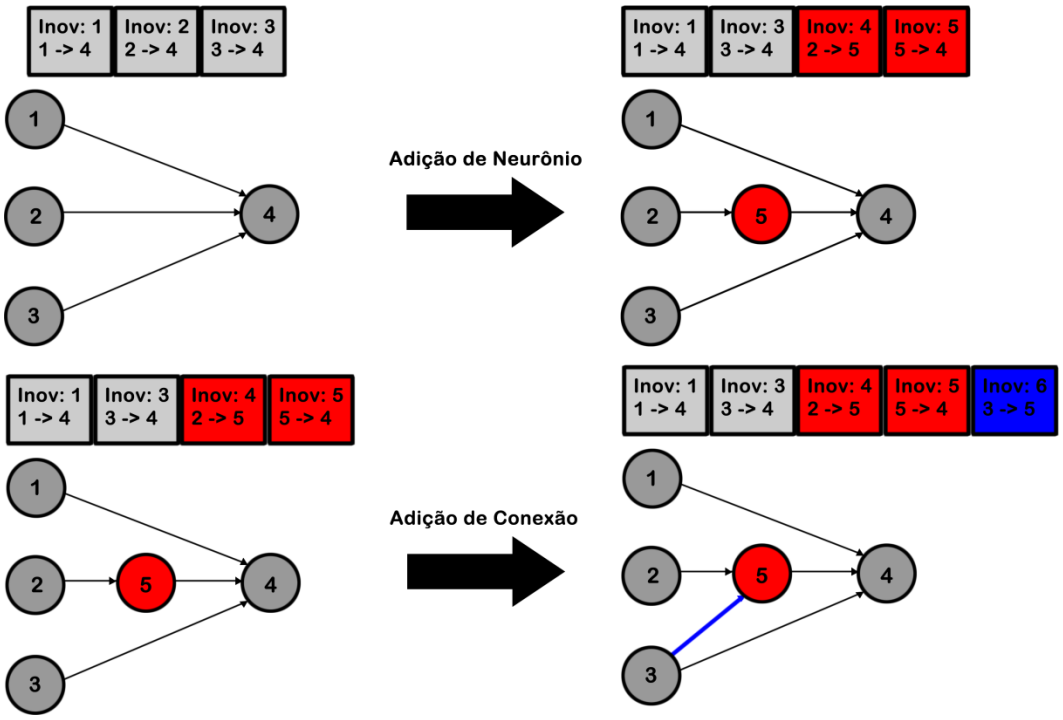

Figura 2.11: Exemplo de mutação estrutural em NEAT.

No método NEAT existem dois tipos de mutação: estrutural e paramétrica. A mutação estrutural é responsável pela adição de neurônios e conexões na topologia da rede neural. Já a mutação paramétrica é mais simples, sendo semelhante aos operadores de mutação tradicionais em que o peso de uma conexão é perturbado aleatoriamente.

A mutação estrutural (Figura 2.11) se divide em duas categorias: adição de conexões e adição de neurônios. A adição de conexões é mais simples, bastando adicionar um novo gene à lista de conexões, conectando dois neurônios previamente não ligados. Já a adição de neurônios é mais complexa, visto que novos neurônios só são adicionados entre conexões já existentes. Assim a conexão existente é eliminada e o novo neurônio é adicionado em seu lugar, bem como duas novas conexões ligando o novo neurônio aos que estavam conectados anteriormente.

2.4.3 Operador de Recombinação

A recombinação em NEAT é mais complexa se comparada aos operadores de recombinação tradicionais, pois deve considerar a topologia da rede neural bem como genótipos de tamanhos diferentes. Esse problema foi minimizado pelo registro histórico de

inovação. Os genótipos dos pais selecionados são alinhados através do registro histórico, sendo que os genes podem ser classificados em três categorias: homólogos, disjuntos e excedidos.

Genes homólogos são os que estão presentes em ambos os pais, genes disjuntos são os que estão presentes em apenas um dos pais e genes excedidos são aqueles presentes em apenas um dos pais que apresenta registro histórico maior do que os genes presentes no outro pai.

Ao fim do alinhamento gênico, genes homólogos são herdados aleatoriamente. Já genes excedidos e disjuntos são herdados apenas do pai com melhor desempenho. Um exemplo do operador de permutação é mostrado na Figura 2.12.

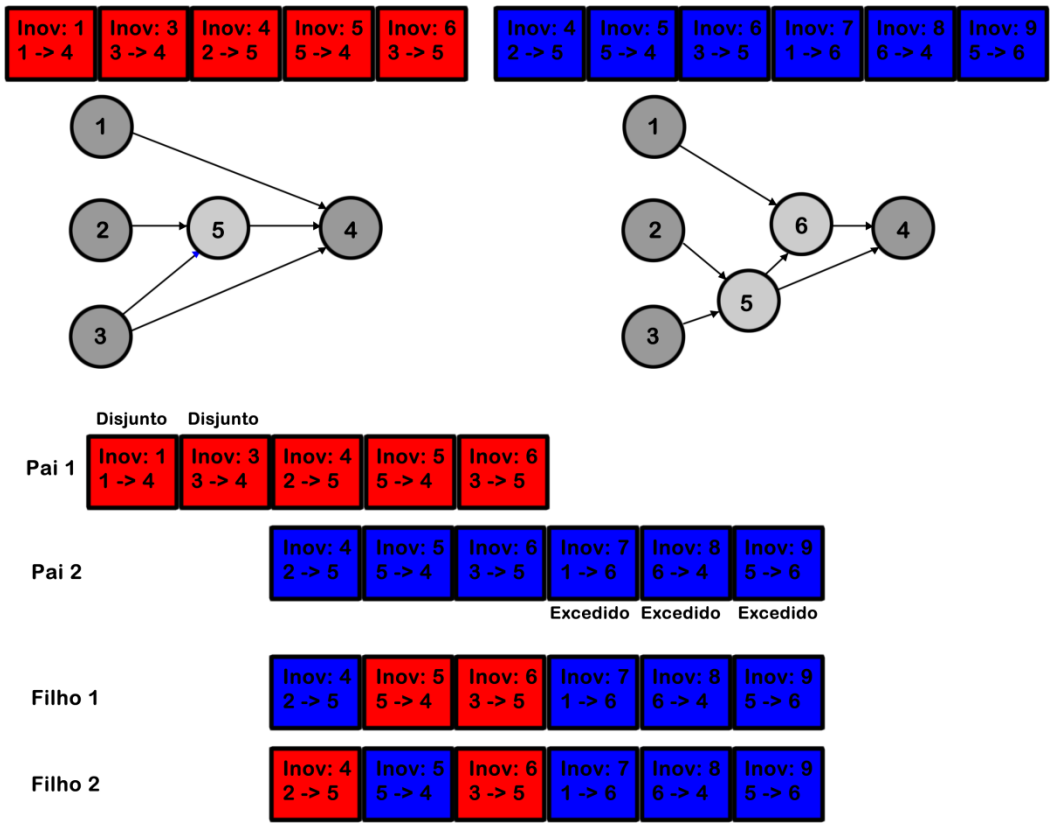

Figura 2.12: Exemplo de operador de permutação.

2.4.4 Especiação, Método de Seleção e Complexificação

A introdução de novos genes e neurônios através dos operadores de mutação pode em um primeiro momento degradar o desempenho da rede neural. Assim, um mecanismo de proteção a inovações é necessário. Esse mecanismo é a especiação. Ela separa indivíduos em

espécies por similaridade através do registro histórico de inovações. Dessa forma, a competição se restringe a indivíduos geneticamente compatíveis, protegendo novas soluções de competições com topologias já otimizadas.

O método de seleção ocorre independentemente em cada espécie. A partir de cada espécie o operador de elitismo é empregado e o indivíduo com melhor desempenho é copiado para a próxima geração, os indivíduos são ordenados conforme seu desempenho, sendo um percentual de indivíduos com pior desempenho descartado e o restante da população escolhida para a reprodução. Na reprodução dois indivíduos são aleatoriamente selecionados e se cruzam para gerar um novo descendente. O processo continua até que o número de indivíduos da nova geração da espécie seja preenchido, o que difere em cada espécie.

Complexificação em NEAT é diferente de outros métodos de neuroevolução. Nesses métodos a primeira geração é inicializada de forma aleatória, dando grande variabilidade gênica para a população, mas desta forma dificilmente se garante que a topologia mínima seja encontrada. Já em NEAT a primeira geração inicia com apenas a camada de entrada conectada à camada de saída, sendo que a complexificação vai ocorrendo a cada geração através dos operadores de mutação e permutação.

2.5 Previsão Numérica

A previsão numérica é uma aplicação da ciência atual que tem como objetivo determinar um estado futuro. Atualmente os sistemas de previsão modernos são compostos de cinco principais componentes (Cintra, 2010): coleta de dados, assimilação de dados, modelo numérico, pós-processamento das saídas do modelo e apresentação da previsão. Este trabalho concentra-se em dois componentes: modelos numéricos e assimilação de dados.

2.5.1 Modelos Numéricos

Os modelos numéricos são simulações computacionais que possuem uma condição inicial. Esses modelos evoluem no tempo empregando conceitos da física do problema. Dois modelos são empregados neste trabalho, o modelo de Água Rasa e o Atractor de Lorenz. Estes modelos foram escolhidos por serem referências da literatura em testes de novas técnicas de assimilação de dados, o modelo de Lorenz por apresentar um comportamento caótico e sensível às condições iniciais já o modelo de Água Rasa é adequado, pois o tamanho

de estado pode ser aumentado arbitrariamente possibilitando testes de desempenho computacional.

2.5.1.1 Atractor de Lorenz

O atractor de Lorenz (Lorenz, 1963) foi introduzido por Edward Lorenz em 1963, que o derivou das equações simplificadas de rodos de convecção que ocorrem nas equações da atmosfera. É um mapa caótico que mostra como o estado de um sistema dinâmico evolui no tempo, em um padrão complexo e não repetitivo. Trata-se de um sistema não linear, tridimensional e determinístico que exibe um comportamento caótico. O Sistema de Lorenz consiste de três equações 1, 2 e 3, diferenciais ordinárias de primeira ordem, acopladas.

$$1.\ \frac{dx}{dt} = -\sigma x - y$$

$$2.\ \frac{dy}{dt} = -\rho x - y - xz$$

$$3.\ \frac{dz}{dt} = xy - \beta z$$

Onde σ, ρ e β são parâmetros do modelo, sendo que as variáveis *x*, *y* e *z* possuem interpretação espacial. O sistema de Lorenz, ilustrado na Figura 2.13, é frequentemente usado em teste de técnicas de assimilação de dados por ser um sistema dinâmico simples com comportamento caótico, sensível às condições iniciais e de fácil implementação.

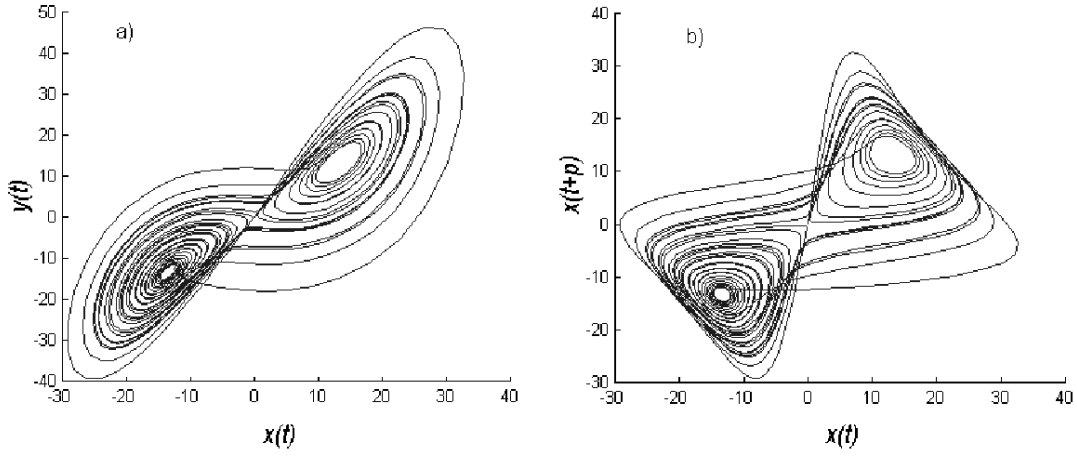

Figura 2.13: Trajetória do sistema de Lorenz em um espaço. (Petry, 2002)

2.5.1.2 Modelo de Água Rasa

As equações de Água Rasa ou *Shallow Water*, também chamadas de equações de Saint Venant (Kranenburg, 1993), são um conjunto de equações diferenciais parciais hiperbólicas que descrevem o fluxo de uma superfície em um fluído. São apropriadas para os fluxos com uma superfície livre e de pequena profundidade, como mostrado na Figura 2.14, podendo ser usadas para descrever o comportamento de um lago ou um rio.

O modelo fornece a evolução da altura do nível da água e a velocidade horizontal. Uma expressão simplificada do modelo é mostrada nas equações 4, 5 e 6.

$$4.\quad \partial_t h + \partial_x(hu) + \partial_y(hv) = 0$$

$$5.\quad \partial_t(hu) + \partial_x(huu) + \partial_y(huv) + \frac{1}{2} g \partial_x h^2 = 0$$

$$6.\quad \partial_t(hu) + \partial_x(huu) + \partial_y(huv) + \frac{1}{2} g \partial_y h^2 = 0$$

O modelo é adequado para o teste de técnicas de assimilação de dados por apresentar continuidade temporal e espacial. O número de variáveis simuladas pode ser definido arbitrariamente, podendo assim ser usado para testes de desempenho computacional.

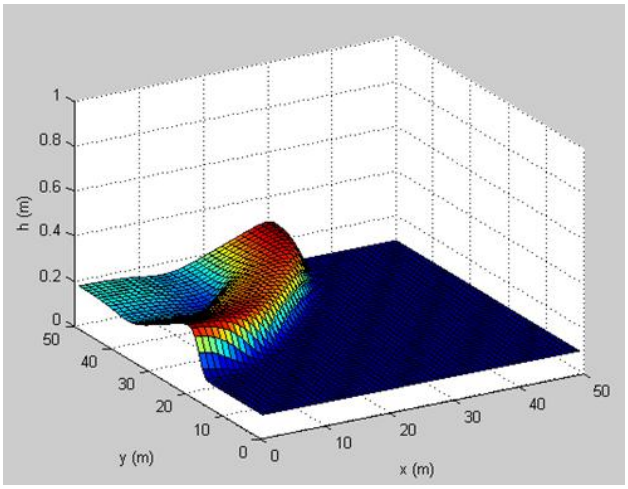

Figura 2.14: Imagens com evolução do modelo de água rasa (Bradford & Sanders, 2002).

2.5.2 Assimilação de Dados

A assimilação de dados é uma forma de manter o estado de um modelo computacional próximo ao real estado da natureza através da assimilação de observações. A análise é a combinação de dados simulados e observações, sendo esse o produto da assimilação de dados.

Na formulação do problema a representação de uma simulação do modelo é denominada vetor de estado x. A melhor representação da realidade é chamando x^t, o estado verdadeiro no momento da análise. O vetor de estado x^b é um estado de referência, obtido através de uma simulação do modelo computacional para o mesmo instante da análise. O vetor x^a é o estado que se pretende encontrar e o produto do processo de assimilação de dados. Assim o problema da análise pode ser definido como uma relação $x^a = x^b + \delta x$, em que x^a deve ser o mais próximo possível de x^t (Cintra & Velho, 2010).

As observações são reunidas em um vetor de observações y. Este vetor precisa ser comparado com o vetor de estado x^b. Assim, uma função de transformação do espaço de modelo para o espaço de observação é necessária, sendo realizada através do operador H. A diferença entre o vetor de observações e o vetor de referência do modelo, $y - H(x^b)$, é denominada vetor de inovação e, quando aplicada à matriz de pesos, gera o incremento da análise.

2.5.2.1 Best Linear Unbiased Estimator

Best Linear Unbiased Estimator ou BLUE é o método fundamental para assimilação de dados, definido pelas equações 7 e 8.

$$7. \quad x^a = x^b + K\left(y - H(x^b)\right)$$

$$8. \quad K = BH^T(HBH^T - R)^{-1}$$

A matriz K representa a função de ganho que é aplicada ao vetor de inovação. Para representar as incertezas sobre o modelo e sobre as observações, são assumidos alguns modelos de erros, calculados a partir de uma função de densidade e probabilidade (FDP) para cada tipo de erro.

Por exemplo, suponha o vetor de background x^b. Existe outro vetor que define seu erro em relação ao real estado da natureza x^t. Esse vetor é dado pela equação 9.a. Obtendo o vetor e_b um grande número de vezes, com as mesmas condições, mas com diferentes erros, gerados por causas desconhecidas podemos calcular dados estatísticos, como média, desvios e histogramas de frequência. Para um número suficientemente grande de vetores de erro, espera-se que os dados estatísticos convirjam para valores que dependam apenas dos processos físicos responsáveis pelos erros. Assim, quando se realizar um novo processo de análise, não se espera saber seu erro, mas sim os dados estatísticos do erro.

A melhor informação sobre o erro é dada pelo limite do histograma quando as classes são infinitamente pequenas e esta é uma função densidade e probabilidade. As FDPs dos erros de observações e do modelo são gaussianas, então x^a é também a estimação de variância mínima de x^t, **B** e **R** são as matrizes de covariância de erros e são calculadas seguindo as equações 9 e 10. Para sistemas reais, como o estado real da natureza x^t é desconhecido, algumas abordagens podem ser aplicadas para estimar as matrizes de covariância **B** e **R**. Uma alternativa é a utilização do método NMC (Kalnay, 2002) onde são utilizadas diferença entre pequenos intervalos de previsão para o mesmo instante de tempo, como por exemplo a diferença entre a previsão de 24 horas com a previsão de 48 horas.

9. **Erros do Modelo:**
   a. $e_b = x^b - x^t$
   b. $\mathbf{B} = \overline{(e_b - \overline{e_b})(e_b - \overline{e_b})^T}$

10. **Erros de Observações:**
   a. $e_0 = y - H(x^b)$
   b. $\mathbf{R} = \overline{(e_0 - \overline{e_0})(e_0 - \overline{e_0})^T}$

3 USANDO O MODELO NEAT EM ASSIMILAÇÃO DE DADOS

3.1 Motivação e Justificativa

A assimilação de dados se faz necessária para a apresentação de previsões que se aproximam ao máximo do verdadeiro estado da natureza, levando-se em consideração o estado de referência x^b, as observações y e o grau de confiança nessas estimativas (matrizes **B** e **R**). No entanto, trazem grandes desafios computacionais. As dificuldades residem na dimensão das matrizes envolvidas nos modelos (Cintra & Velho, 2010), atualmente matrizes cheias da ordem de 10^{12} elementos.

Nestes cenários a aplicação de redes neurais em assimilação de dados vem sendo proposta em vários trabalhos recentes (Cintra & Velho, 2010), (Cintra, Velho, & Todling, 2009), (Härter, 2004), (Nowosad, 2001), (Liaqat, Fukuhara, & Takeda, 2001), (William, Hsieh, & Tang, 1998) e (Yu, Iredell, & Keyser, 1997) como solução desse desafio. Nestas propostas, redes neurais são aplicadas como emuladores dos métodos clássicos de assimilação de dados reduzindo significativamente o custo computacional.

3.2 Metodologia e Contribuição

O problema de assimilação de dados pode ser visto resumidamente em uma equação $x^a = x^b + d$, onde d representa o incremento da análise. O vetor de incremento é calculado como produto da matriz de ganho pelo vetor de diferenças entre os vetores de *background* e de observações, $d = K(y - x^b)$. Sendo K a matriz de ganho, que é obtida por métodos clássicos de assimilação de dados, estes métodos produzem a matriz considerando erros de modelo e observacionais. A rede neural emula a matriz de ganho como definido a seguir, $d = R_{rn}(y - x^b)$. Assim, tem-se por objetivo usar uma rede neural para emular o uso de um método clássico de assimilação de dados, tirando proveito do significativo ganho de desempenho computacional.

Os trabalhos na área diferem principalmente em relação ao modelo em que se esta empregando a técnica de assimilação de dados. Com exceção da tese de doutorado de Härter (Härter, 2004), todos os trabalhos verificados empregam uma rede neural direta com topologia fixa treinada por *backpropagation*.

A tese de Härter propõe a verificação do desempenho do uso de outros modelos de rede neurais, todos de topologia fixa. Os modelos empregados são: rede neural direta; rede neural de função de base radial, que difere da rede neural tradicional por possuir uma função de base radial como função de ativação; rede de Elmam e rede de Jordan que possuem diferentes tipos de conexões recorrentes, aplicadas aos modelos de Lorenz e Água Rasa e treinadas através *backpropagation*. As redes neurais recorrentes apresentaram na fase de treinamento um desempenho computacional superior as redes neurais direta e de base radial, pois foi necessário um número significativamente menor de iterações para o treinamento. Além disso, também foram capazes de resolver o problema de assimilação de dados com uma qualidade semelhante a das redes neurais direta e de função de base radial, mas com um menor número de neurônios. Sendo que um menor número de neurônios implica em um melhor desempenho computacional no uso do processo de assimilação de dados.

O trabalho de Härter mostrou que é possivel aumentar o desempenho computacional com o uso de conexões recorrentes. No entanto nenhum trabalho verificado explorou uma das caracteristicas mais importantes de uma rede neural (Miguel, 2009), a topologia, pois com uma topologia minima como mostrado no trabalho de (Stanley & Miikkulainen, 2002) pode-se aumentar significativamente o desempenho de uma rede neural.

Assim esse trabalho pretende demonstrar a aplicação de um novo modelo de neuroevolução objetivando ganho de desempenho computacional. As contribuiçoes deste trabalho residem em duas principais linhas. Primeiramente ao propor uma aplicação de um novo modelo de rede neural ao problema de assimilação de dados objetivando ganho de desempenho computacional, algo muito relevante ao problema proposto. A segunda contribuição reside no desenvolvimento do software de testes que possibilite a verificação da eficiência e eficácia desse novo modelo de rede neural.

3.3 Desenvolvimento

Neste capítulo serão descritos os principais módulos do protótipo de testes, bem como conceitos e tecnologias empregadas. O software desenvolvido tem como objetivo oferecer uma plataforma de testes, em que se possa constatar a eficácia ou não do método de neuroevolução NEAT.

3.3.1 Processo de Desenvolvimento

Inicialmente um protótipo do software foi desenvolvido em C++ onde todos os componentes foram implementados a partir dos referenciais teóricos. Após a verificação dos resultados iniciais, optou-se pelo de uso de bibliotecas padrões, visto que dessa forma uma análise mais independente de implementação poderia ser realizada.

A partir desse momento foi realizada uma analise para determinação da linguagem de programação que seria utilizada. Inicialmente foram consideradas todas as linguagens que possuíam uma implementação do método NEAT, sendo elas: C++, Java, Delphi, Matlab e C#. Todas essas linguagens possuíam bibliotecas satisfatórias para operações algébricas e de redes neurais gerais. Assim, a escolha da linguagem foi determinada apenas pelo nível da implementação do método NEAT já que muitas delas se apresentam como protótipos experimentais e não como frameworks estáveis. A avaliação do nível de implementação foi realizada pelo numero de publicações associada a implementação. Dessa forma, a linguagem escolhida foi C#, pois o framework SharpNEAT é o que possuía o maior numero de publicações atreladas a ele.

3.3.2 Estrutura Geral do Protótipo de Testes

O protótipo desenvolvido possui seis principais pacotes: **PROTOTYPE**, **EXPERIMENTS**, **MODELS**, **METHODS**, **DATASET** e **NEURAL NETWORKS**. O Grafo de dependência de uma execução do protótipo é exemplificado na figura 3.1 Na figura a largura da seta é um fator relacionado ao nível de dependência dos pacotes.

Uma execução inicia pela invocação do pacote **PROTOTYPE** que possui o método **MAIN** do programa. Após o pacote **MODELS** é invocado selecionando o modelo que será usado no experimento, em seguida o pacote **EXPERIMENTS** é acionado em conjunto com o pacote **NEURAL NETWORKS** adicionando os modelos de redes neurais que serão empregadas. Por fim, o método **RUN** do pacote **EXPERIMENTS** é acionado iniciando o processo de experimentação.

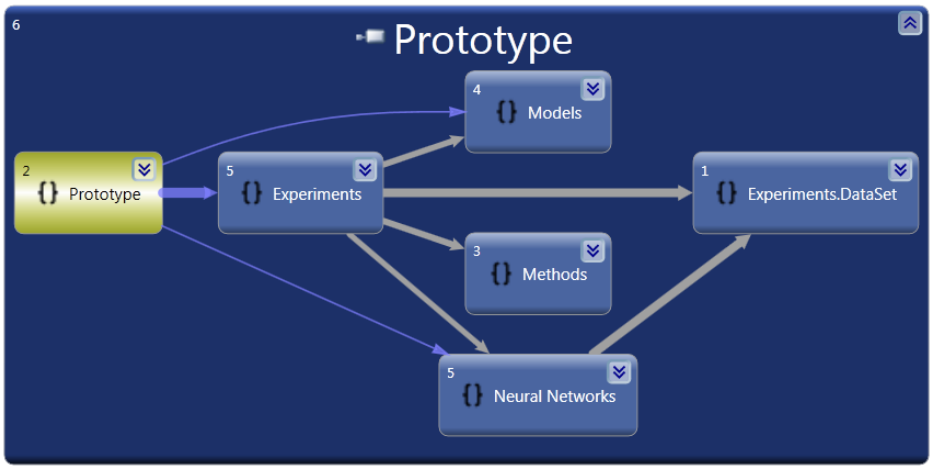

Figura 3.1: Estrutura de execução do software de testes.

A figura 3.2 exemplifica o código necessário no método **MAIN** para uma execução completa do protótipo. Na primeira linha são definidos parâmetros comuns a todos os modelos de redes neurais: população, número de iterações e número de neurônios na camada oculta. Em seguida é fornecido ao **EXPERIMETER** o modelo que será usado através do método **FACTORY** da classe **MODEL**. Após é definido o tipo de experimento que será realizado: treinamento, validação ou previsão. Neste momento também são adicionados os modelos de redes neurais que serão empregadas no experimento. O último método a ser invocado é o método **RUN** que inicia a execução do experimento.

Toda a estrutura do software foi desenvolvida utilizando-se largamente boas práticas de programação bem como padrões de programação. O protótipo foi desenvolvido com objetivo de permitir que a realização dos experimentos seja a mais simples possível, já que foi necessário a realização de um grande número de experimentos para a determinação dos vários parâmetros relacionados a esse trabalho.

```
class Program
{
    static void Main(string[] args)
    {
        NeuralNetwork.SetParameters(100, 1000, 6);
        Experimenter expter = new Experimenter(Model.Factory(ModelType.Lorenz));
        Experiment exp = new ExperimentTrain();

        exp.addNeuralNetwork(NeuralNetwork.Factory(NeuralNetoworkType.BackPropagation));
        exp.addNeuralNetwork(NeuralNetwork.Factory(NeuralNetoworkType.EvolutionaryLearning));
        exp.addNeuralNetwork(NeuralNetwork.Factory(NeuralNetoworkType.NEAT));
        exp.addNeuralNetwork(NeuralNetwork.Factory(NeuralNetoworkType.HyperNEAT));
        expter.AddExperiment(exp);

        exp = new ExperimentValidation();

        exp.addNeuralNetwork(NeuralNetwork.Factory(NeuralNetoworkType.BackPropagation));
        exp.addNeuralNetwork(NeuralNetwork.Factory(NeuralNetoworkType.EvolutionaryLearning));
        exp.addNeuralNetwork(NeuralNetwork.Factory(NeuralNetoworkType.NEAT));
        exp.addNeuralNetwork(NeuralNetwork.Factory(NeuralNetoworkType.HyperNEAT));
        expter.AddExperiment(exp);

        exp = new ExperimentForecast();

        exp.addNeuralNetwork(NeuralNetwork.Factory(NeuralNetoworkType.BackPropagation));
        exp.addNeuralNetwork(NeuralNetwork.Factory(NeuralNetoworkType.EvolutionaryLearning));
        exp.addNeuralNetwork(NeuralNetwork.Factory(NeuralNetoworkType.NEAT));
        exp.addNeuralNetwork(NeuralNetwork.Factory(NeuralNetoworkType.HyperNEAT));
        expter.AddExperiment(exp);

        expter.Run();
    }
}
```

Figura 3.2: Exemplo de código para o método **MAIN** para uma execução completa do programa de testes.

3.3.3 Pacote Propotype

O pacote **PROTOTYPE** é o pacote mais simples do software. Ele possui apenas duas classes chamadas **PROGRAM** e **UTILS**. A classe **PROGRAM** possui o método **MAIN** sendo esse o método inicial do programa. Já a classe **UTILS** possui várias sobrecargas do método **CALCERROR** utilizado em várias situações no software (Figura 3.3).

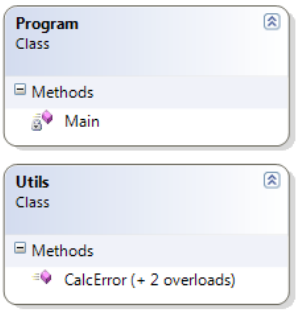

Figura 3.3: Classes pertencentes ao pacote **PROTOTYPE**.

3.3.4 Pacote Models

O pacote **MODELS** é responsável por realizar a iteração ou carregamento para obtenção dos dados dos modelos físicos. Nele são implementados os modelos de Lorenz e a classe responsável pelo carregamento dos dados do modelo Água Rasa. O pacote possui três principais classes: **MODEL**, **LORENZ** e **SHALLOWWATER** (Figura 3.4).

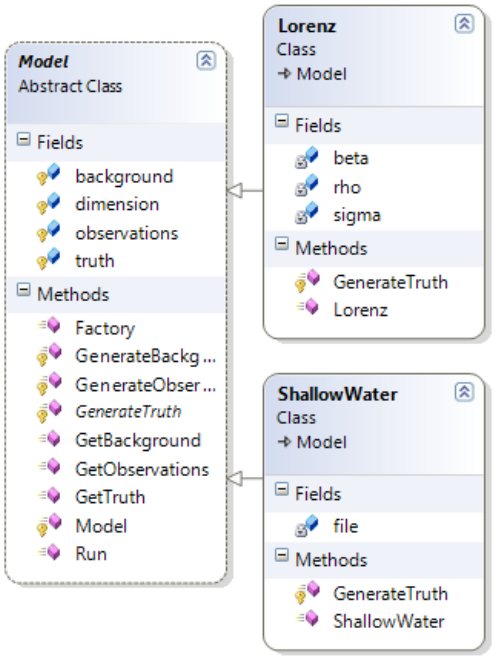

Figura 3.4: Principais classes pertencentes ao pacote **MODELS**.

A classe **MODEL** é uma classe abstrata que implementa todos métodos necessários para seu uso, delegando a apenas a implementação do método **GENERATETRUTH** às classes que a estendem. Através da interface disponibilizada pela classe **MODEL** pode-se operar sobre diferentes modelos físicos com alto nível de abstração. Nela são usados dois padrões de projeto: **FACTORY METHOD** e **TEMPLATE METHOD** (Gamma, Helm, Johnson, & Vlissides, 1994). O padrão **FACTORY** é um padrão criacional empregado no sentido de ocultar detalhes de criação de cada tipo de modelo físico, facilitando assim o uso da classe. O padrão **TEMPLATE METHOD** define o esqueleto de um algoritmo, postergando a definição de alguns passos do algoritmo as para subclasses. Assim o método **RUN** define o algoritmo para geração dos dados de *background* e das observações, delegando a criação do conjunto verdade à classe que a estende (Figura 3.5).

```
//Design Pattern - Template Method
public void Run()
{
    GenerateTruth();
    GenerateObservations();
    GenerateBackground();
}

protected abstract void GenerateTruth();
protected void GenerateObservations()...
protected void GenerateBackground()...
```

Figura 3.5: Implementação do padrão de projeto **TEMPLATE METHOD**.

A classe **LORENZ** estende a classe **MODELS** e é responsável por gerar todo o conjunto verdade através da implementação do método **GENERATETRUTH**. O modelo foi implementado neste trabalho utilizando os mesmo parâmetros da tese de doutorado de Cintra (Cintra R. S., 2010).

A classe **SHALLOWWATER** estende a classe **MODELS** e é responsável por carregar todo o conjunto verdade através da implementação do método **GENERATETRUTH**. O modelo de Água Rasa possui uma implementação complexa, envolvendo conceitos avançados sobre técnicas de métodos numéricos e volumes finitos. Neste trabalho optou-se por não implementar o modelo, já que ele possui validação complexa. Assim, os dados para os experimentos foram obtidos através da execução do código em Matlab, atrelado ao trabalho de (Bradford & Sanders, 2002), disponível em (Bradford & Sanders, 2011). Assim a classe **SHALLOWWATER** se limita ao carregamento dos dados previamente gerados.

3.3.5 Pacote Methods

O pacote **METHODS** é responsável por realizar a assimilação de dados. Nele é implementado o método *Best Linear Unbiased Estimator*. A implementação foi realizada com base nas equações disponíveis em (Bouttier & Courtier, 1999). De forma geral o pacote recebe os dados oriundos do modelo físico e retorna a análise resultado do processo de assimilação de dados.

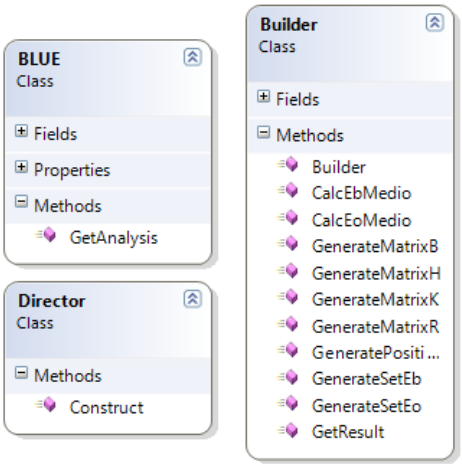

Figura 3.6: Principais classes pertencentes ao pacote **METHODS**.

O método BLUE é representado pela classe de mesmo nome. A construção desse objeto é complexa e exige que sejam geradas uma grande quantidade de matrizes. O padrão

de projeto indicado na construção de objetos complexos é o **BUILDER** (Gamma, Helm, Johnson, & Vlissides, 1994), sendo este padrão empregado na construção do objeto BLUE.

O padrão de projeto implementado utiliza três classes **DIRECTOR**, **BUILDER** e **BLUE**. A classe **BUILDER** especifica uma interface para um construtor de partes do objeto-produto, já a classe **DIRECTOR** constrói o objeto-produto utilizando a interface disponibilizada pela classe **BUILDER**. Por fim a classe **BLUE** é o objeto-produto resultado da construção.

Para a implementação do método BLUE uma série de operações matriciais e vetoriais são necessárias, tais como: multiplicações, transposições e inversões. Já que essas operações são utilizadas constantemente, necessitam de grande eficiência. Optou-se por utilizar a biblioteca numérica *Math.NET Numerics* (Rüegg, Cuda, & Gael, 2011). Esta biblioteca fornece métodos e algoritmos para cálculos numéricos em ciência e engenharia, cobrindo tópicos como funções especiais, álgebra linear, modelos de probabilidade, números aleatórios, interpolação, transformadas integrais (FFT).

3.3.6 Pacote Neural Networks

O pacote **NEURAL NETWORKS** é responsável por realizar a emulação da técnica de assimilação de dados. O pacote é composto por três principais classes: **NEURALNETWORK**, **NEURALNETWORKNEAT** e **NEURALNETWORKAFORGE**.

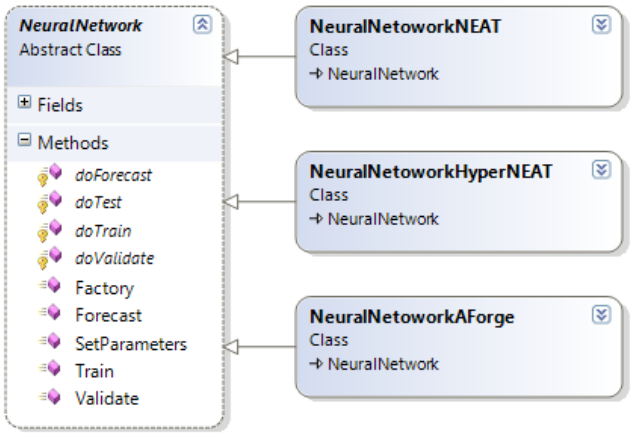

Figura 3.7: Principais classes pertencentes ao pacote **NEURAL NETWORKS**.

A classe **NEURALNETWORK** é uma classe abstrata que implementa a estrutura geral de uma rede neural usada no protótipo. A partir da interface disponibilizada pela classe abstrata pode-se operar sobre diferentes redes neurais com um alto nível de abstração. A classe emprega três

padrões de projeto **FACTORY METHOD**, **TEMPLATE METHOD** e **STRATEGY** (Gamma, Helm, Johnson, & Vlissides, 1994).

O padrão **FACTORY** é empregado no sentido de ocultar detalhes de criação de cada modelo de rede neural, como mostrado na figura 3.7 uma rede neural é criada simplesmente especificando seu tipo através do método **FACTORY**. O padrão **TEMPLATE METHOD** é usado na exibição dos resultados de cada tipo de experimento nos métodos **TRAIN**, **FORECAST** e **VALIDATE**. Através desse padrão a exibição é especificada uma única vez para cada experimento, sem qualquer replicação de código. Um exemplo de sua aplicação pode ser observado na figura 3.8.

```
abstract protected double doForecast();
public void Forecast()
{
    System.Console.WriteLine(name + " Network - Forecast:");

    System.Console.WriteLine("Error: {0}", doForecast());
    System.Console.WriteLine();
}
```

Figura 3.8: Exemplo do padrão de projeto **TEMPLATE METHOD** aplicado ao método **FORECAST**.

Já o padrão de projeto **STRATEGY** permite definir algoritmos em tempo de execução, sem alterar a classe do elemento sobre o qual opera. Este padrão de projeto é implementado através da classe **NEURALNETWORKAFORGE**, ela estende a classe **NEURALNETWORK** e define todas a operações necessárias para o seu uso através das classes oferecidas pelo pacote AForge.NET. Assim as redes neurais treinadas por *backpropagation* e algoritmos genéticos são implementadas por uma única classe sendo definido em tempo de execução o algoritmo de treinamento utilizado.

As redes neurais diretas treinadas por *backpropagation* e algoritmos genéticos foram implementadas através da biblioteca AForge.NET (AForge.NET, 2011). Esta biblioteca consiste de um framework desenvolvido em C# para desenvolvedores e pesquisadores nas áreas de Visão Computacional e Inteligência Artificial - processamento de imagens, redes neurais, algoritmos genéticos, aprendizado de máquina, a robótica e etc. A biblioteca é composta de uma serie de sub-pacotes: AForge.Imaging, AForge.Vision, AForge.Video, AForge.Neuro, AForge.Genetic, AForge.Fuzzy, AForge.Robotics e AForge.MachineLearning.

Neste trabalho foram usado os pacotes AForge.Neuro e AForge.Genetic. Estes pacotes contêm todas as classes e métodos necessários para o uso de ambas as formas de treinamento

das redes neurais. Seu uso se dá pela invocação de classes e especificação de parâmetros, não sendo necessário nenhum esforço maior de programação.

A classe **NeuralNetoworkNEAT** estende a classe **NeuralNetwork** e implementa todos os métodos necessários para o seu uso. Para isso, utiliza duas outras classes **NEATExperiment** e **NEATEvaluator**. Essas classes são necessárias para se utilizar o framework SharpNEAT. A classe **NEATExperiment** estende a classe **INeatExperiment** e define um experimento de acordo com o framework fixado pelo pacote SharpNEAT. Já a classe **NEATEvaluator** define uma forma de avaliar a redes neurais definidas pela classe **NEATExperiment** (Figura 3.9).

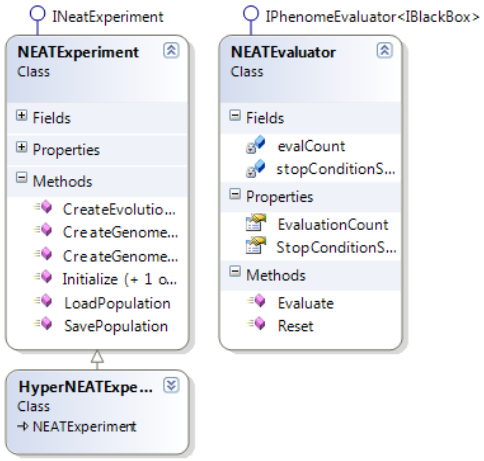

Figura 3.9: Classes necessárias para utilizar o framework SharpNEAT.

O pacote SharpNEAT (SharpNEAT , 2011) é um framework otimizado de neuroevolução escrito em C# que contém toda a implementação-base para experimentos utilizando NEAT. Seu uso é complexo e exige a extensão de classes e definição de experimentos. Apesar de ser utilizada em vários trabalhos, não existe documentação para o framework.

3.3.7 Pacote DataSet

O pacote **DataSet** possui duas principais atribuições que são definir um ponto único de acesso global a todos os dados utilizados nos experimentos e gerar a partir dos dados fornecidos pelo pacote **Model** os conjuntos de dados utilizados no treinamento e avaliação dos modelos de redes neurais.

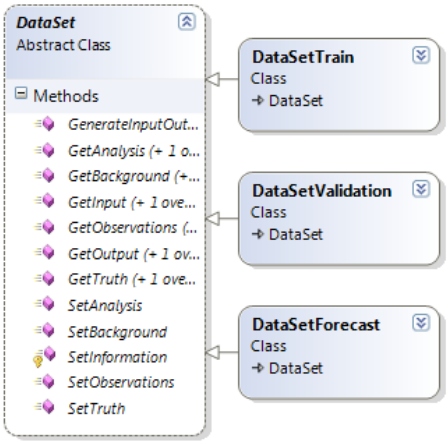

Figura 3.10: Principais classes pertencentes ao pacote **DATASET**.

A primeira atribuição é suprida através do uso do padrão de projeto **SINGLETON** (Gamma, Helm, Johnson, & Vlissides, 1994). Este é um padrão criacional que garante que uma classe tenha apenas uma instância e fornece um ponto global de acesso a ele.

O pacote é composto pela classe abstrata **DATASET** que fornece uma interface genérica de acesso aos tipos específicos que estendem esta classe, e permite que seja possível operar sobre eles com alto nível de abstração. O pacote também é composto por **DATASETTRAIN**, **DATASETVALIDATION** e **DATASETFORECAST** sendo cada um responsável por gerar um conjunto de dados diferente.

3.3.8 Pacote Experiments

Todos os pacotes descritos até agora são de fato utilizados pelo pacote **EXPERIMENTS** através de suas classes. A arquitetura do pacote **EXPERIMENTS** foi projetada de acordo com o padrão de projeto **DECORATOR** (Gamma, Helm, Johnson, & Vlissides, 1994). Este padrão permite que sejam adicionadas reponsabilidades a um objeto dinamicamente (Figura 3.11).

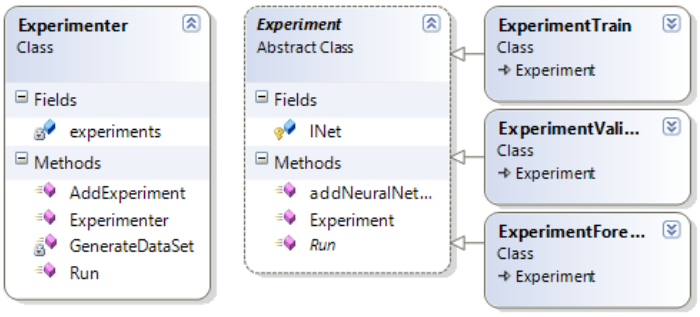

Figura 3.11: Principais classes pertencentes ao pacote **EXPERIMENT**.

Em um primeiro momento os modelos de redes neurais são adicionados a objetos de classes que estendem a classe **EXPERIMENT.** Após esses objetos são adicionados a classe **EXPERIMENTER**. O uso desses vários níveis de abstração permite que execução dos experimentos ocorra de forma simples, como mostrado na figura 3.12.

```
class Experimenter
{
    ...

    public void Run()
    {
        foreach (Experiment ex in experiments)
        {
            ex.Run();
        }
    }
}
class ExperimentTrain : Experiment
{
    override public void Run()
    {
        ....
        foreach (NeuralNetwork net in lNet)
        {
            ...
            net.Train();
            ....
        }
    }
}
```

Figura 3.12: Exemplo de execução de um experimento de treino.

4 RESULTADOS

Neste capítulo serão descritos os resultados deste trabalho obtidos através da execução do software desenvolvido.

4.1 Ambiente de Teste

Os experimentos foram executados em um servidor com a seguinte configuração: 2 processadores Intel Xeon CPU E5620 2.40 GHz com 16 núcleos de processamento e 32 GB de memória RAM. Uma execução do protótipo de testes é completamente serial, assim quantidade de processadores disponível não afeta o resultado experimento. O poder computacional disponível foi usado no sentido de possibilitar a execução de vários experimentos simultaneamente, já que muitos parâmetros precisam ser ajustados, o que implica em muitas execuções para valida-los.

4.2 Caso de Teste: Atractor de Lorenz

O caso do modelo de Lorenz será empregado no sentido de avaliar a capacidade de aprendizagem dos modelos neuroevolucionistas quando comparados com a uma rede neural treinada por *backpropagation*. O comportamento caótico do modelo apresenta um desafio para as técnicas de aprendizagem, assim segundo a literatura é valido afirmar que uma rede neural consegue emular um método de assimilação de dados, se for capaz emular o método de assimilação de dados para o Atrator de Lorenz.

A fim de possibilitar uma comparação com os trabalhos realizados por (Härter, 2004) e (Nowosad, 2001) empregou-se os mesmo parâmetros e métodos para a realização dos experimentos. O modelo de Lorenz foi integrado com $\Delta t = 10^{-3}$, para $X(0) = 1.508870$, $Y(0) = -1.531271, Z(0) = 25.46091$, $\sigma = 10.0, r = 8.0/3.0$ e $b = 28.0$, ou seja, parâmetros com os quais a dinâmica do modelo segue uma trajetória caótica.

As redes neurais foram treinadas com 2000 exemplos, sendo 400 dados usados na validação cruzada. Esta quantidade de padrões foi determinada empiricamente por (Härter, 2004) e (Nowosad, 2001), sendo considerada uma quantidade de padrões necessária e suficiente para que a rede neural possa aprender. O melhor conjunto de generalização foi fixado em 4000 épocas de treinamento.

Após o treinamento e validação, as redes neurais são submetidas ao teste de previsão de 10^6 passos. Essa previsão de longo prazo foi empregada por (Härter, 2004) para verificação da capacidade de generalização das redes neurais com conexões recorrentes.

As redes neurais de topologia fixa treinadas por *backpropagation* e algoritmos genéticos foram especificadas segundo a literatura, com seis neurônios na camada oculta e três na camada de saída.

Os 400 dados usados na validação cruzada foram considerados, segundo (Härter, 2004), uma quantidade suficiente para analisar graficamente se a rede neural está conseguindo generalizar, e assim obter a melhor estimativa com os dados não pertencentes ao conjunto de treinamento. Os erros de treinamento (ET), validação cruzada (EV) e previsão (EP) são dados por:

$$ET = \frac{1}{2000}\sum_{i=1}^{2000}\frac{1}{2}\sqrt{(x_i^{alvo} - x_i^{rn})^2}$$

$$EV = \frac{1}{400}\sum_{i=1}^{400}\sqrt{(x_i^{alvo} - x_i^{rn})^2}$$

$$EP = \frac{1}{10^6}\sum_{i=1}^{10^6}\sqrt{(x_i^{alvo} - x_i^{rn})^2}$$

Os erros de treinamento, validação cruzada e de previsão resultantes do experimento computacional são apresentados a seguir na figura 4.1.

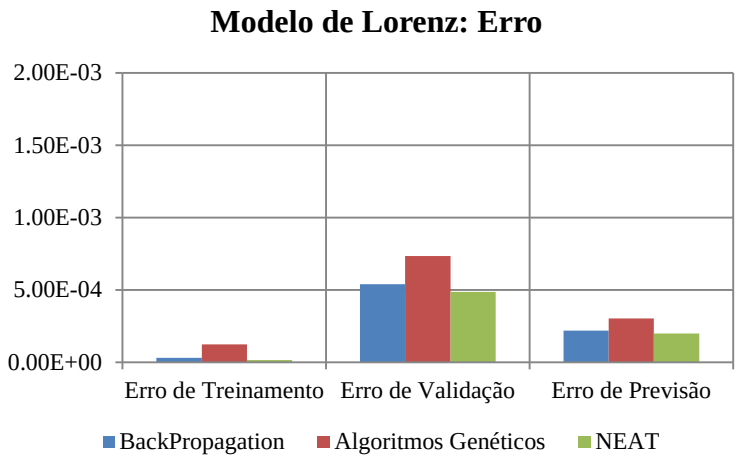

Figura 4.1: Gráfico dos erros do experimento computacional para o atrator de Lorenz.

Na figura 4.1 é possível observar que todos os métodos foram capazes de emular o método de assimilação de dados com base no conjunto de treinamento, sendo que método NEAT obteve o menor erro para os três conjuntos de dados. Assim através do modelo de Lorenz pode-se verificar que todos os modelos de rede neural são eficazes quanto a sua aplicação.

As redes neurais treinadas por *backpropagation* e algoritmos genéticos foram fixadas com seis neurônios na camada oculta e três na camada de saída, especificadas segundo a literatura. Essa configuração resulta em um total de 63 operações de multiplicação a cada *update* da rede neural. Já a rede neural treinada por NEAT encontrou uma topologia de três neurônios na camada oculta e três na camada de saída, com um total de 13 conexões o que implica diretamente em 13 operações de multiplicação a cada *update* da rede neural. Esses resultados podem ser observados na figura 4.2.

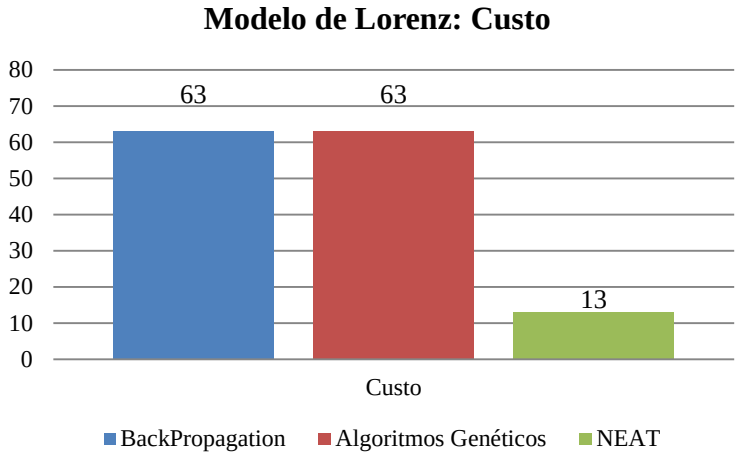

Figura 4.2: Custo em número de operações de multiplicação para uma operação de *uptade*.

Assim pode-se observar que o método NEAT foi capaz de resolver o problema proposto com um custo computacional de significativamente menor quando comparada a solução das redes neurais de topologia fixa e ainda com uma menor taxa de erro para todos os conjuntos de treinamanto.

4.3 Caso de Teste: Modelo de Água Rasa

O caso do modelo de Água Rasa será empregado no sentido de avaliar a capacidade de ganho computacional dos modelos neuroevolucionistas quando comparados com a rede neural treinada por *backpropagation*. A partir desse momento serão considerados apenas a rede neural treinada por *backpropagation* e o modelo NEAT. O modelo de Água Rasa é adequado para o teste de técnicas de assimilação por apresentar continuidade temporal e espacial. Como já mencionado neste trabalho o número de variáveis simuladas pode ser definido arbitrariamente, podendo assim ser usado para testes de desempenho computacional. Foram considerados quatro casos de testes com grids de 5x5, 10x10, 15x15 e 20x20, o que corresponde respectivamente a estados de modelo com 25, 100, 225 e 400 variáveis.

O processo usado é exatamente o mesmo empregado no experimento do modelo de Lorenz. As redes neurais foram treinadas com 2000 exemplos, sendo 400 dados são usados na validação cruzada. O melhor conjunto de generalização foi fixado em 4000 épocas de treinamento. Após o treinamento e validação, as redes neurais são submetidas ao teste de previsão de 10^6 passos.

A rede neural de topologia fixa treinada por *backpropagation* foi especificada de acordo com o melhor desempenho no conjunto de validação. Considerando a suavidade do modelo de Água Rasa a rede neural obteve o melhor desempenho com nenhum neurônio na camada oculta, obtendo-se assim uma rede neural simples com apenas a camada de saída, sendo que essa é a configuração com melhor desempenho computacional para uma rede neural totalmente conectada.

Nas figuras 4.3, 4.4 e 4.5 são apresentados os erros provenientes dos experimentos com os diferentes tamanhos de grid.

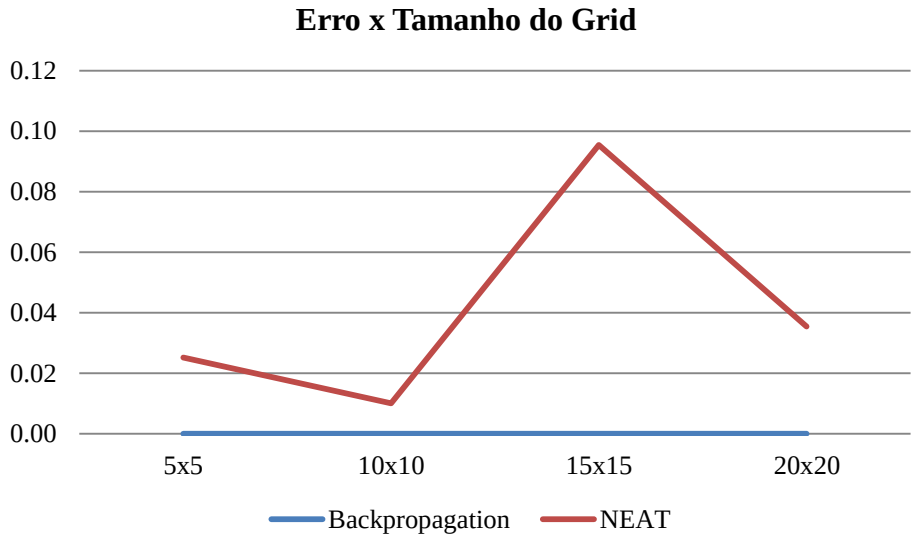

Figura 4.3: Erro para o conjunto de treinamento para os diferentes tamanhos de grid.

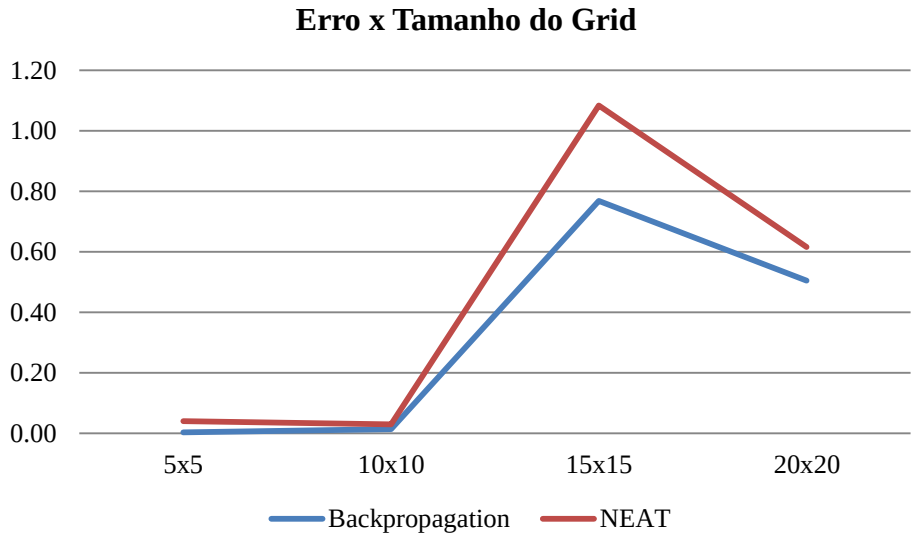

Figura 4.4: Erro para o conjunto de validação para os diferentes tamanhos de grid.

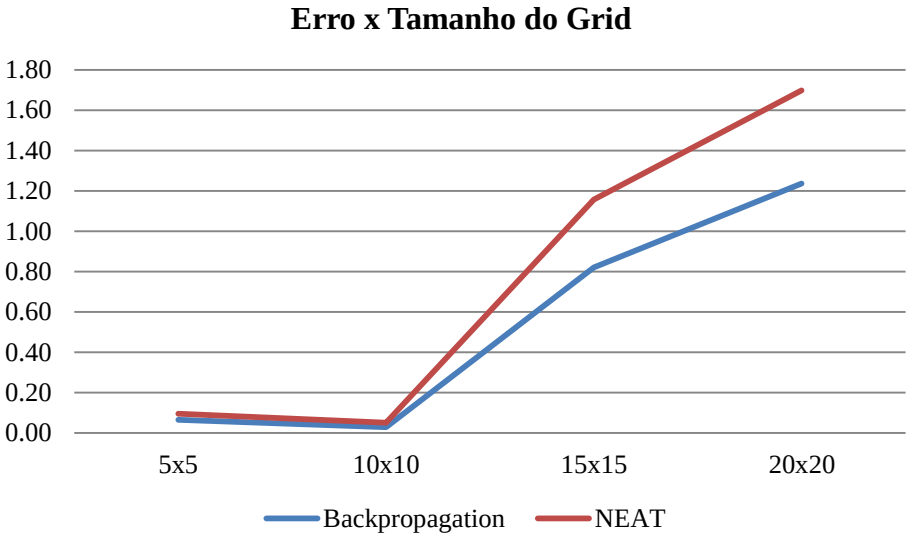

Figura 4.5: Erro para o conjunto de previsão para os diferentes tamanhos de grid.

A partir das figuras 4.3, 4.4 e 4.5 é possível verificar que a rede neural treinada por *backpropagation* sempre obteve os melhores desempenhos em relação ao erro. Para o conjunto de treinamento a rede neural treinada por *backpropagation* foi muito superior a rede neural treinada pelo método NEAT. No entanto para os conjuntos de validação e previsão a diferença entre o método NEAT e a rede neural treinada por *backpropagation* foi reduzida de forma significativa, mostrando que o método NEAT é capaz de generalizar de forma eficaz, que é essencial para ser usado como método alternativo de assimilação de dados.

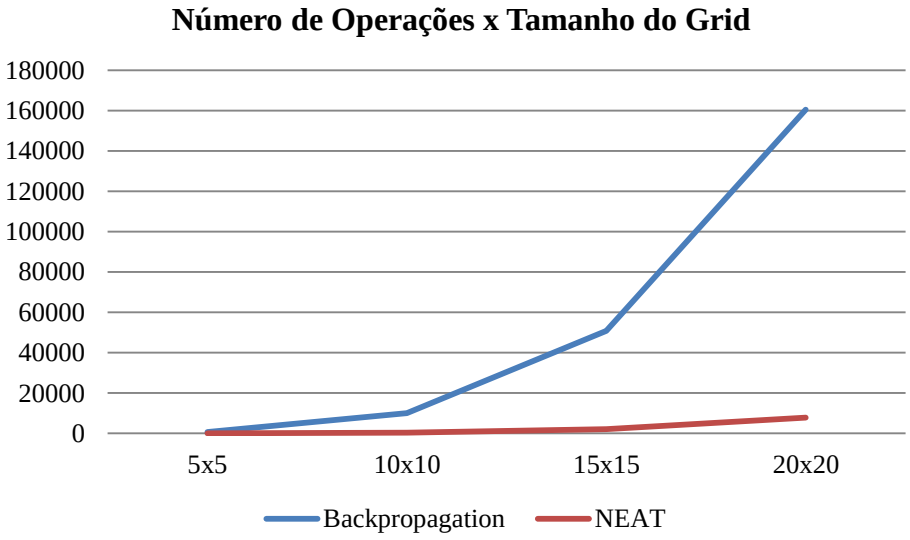

Figura 4.6: Custo em número de operações de multiplicação para execução de uma operação de *update*.

A figura 4.6 exibe um gráfico com o número de operações de multiplicação necessárias para realização de uma operação de *update* para cada tamanho de grid. É possível verificar que a rede neural treinada pelo método NEAT foi muito superior nesse quesito. A topologia ótima apresentada pelo método NEAT apresentou uma tendência de crescimento linear em relação ao tamanho do grid. Já a rede neural totalmente conectada, mesmo sem possuir camada oculta, demonstrou uma tendência de crescimento mais acentuada no número de operações necessárias.

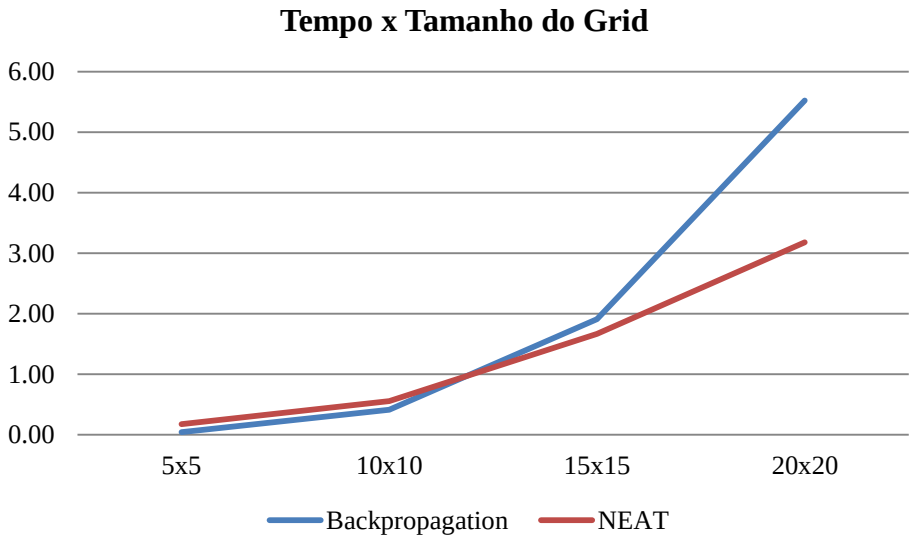

Figura 4.7: Tempo em segundos para realização de dez mil operações de *update*.

A figura 4.7 apresenta os resultados dos experimentos em relação ao tempo em segundos para efetuar a assimilação de dados para todo o conjunto de previsão. Dado o pequeno tempo dos experimentos, os resultados consistem da média da repetição de dez vezes o experimento.

O método NEAT mesmo mostrando-se superior em número de operações para todos os casos de testes, não mantém o mesmo desempenho quando o tempo computacional é considerado. Assim para os casos com grid de 5x5 e 10x10 NEAT apresenta um pior desempenho, mas ainda muito próximo da rede totalmente conectada. Isso é natural, pois o método NEAT apresenta uma representação mais complexa com varias camadas de abstração. Em contrapartida a rede neural totalmente conectada é representada através de matrizes fazendo com que uma operação de *update* seja resolvida de forma muito eficiente. Mas para

os estados maiores, com o aumento da vantagem do método NEAT em relação a numero de operações, este também começa a apresentar uma vantagem em relação ao tempo computacional. Assim, mesmo com uma representação mais complexa o método NEAT pode apresentar um melhor desempenho computacional, principalmente em problemas com estados maiores.

5 CONCLUSÃO

Este trabalho apresentou uma nova abordagem para realização de assimilação de dados, utilizando o conceito de topologia mínima através do método NEAT. Os dados provenientes nos experimentos são obtidos através dos modelos de Loren e Água Rasa, ambos referência em testes de novas técnicas de assimilação de dados. Três modelos de redes neurais foram empregas nos experimentos: dois modelos de redes neurais de topologia fixa treinadas por backpropagation e por algoritmos genéticos e o terceiro método neuroevolucionista NEAT. O método BLUE (Figura 6.1) é o método de assimilação de dados a ser emulado nos experimentos.

Foram realizados cinco experimentos verificando eficácia e eficiência na tarefa de emulação de técnicas de assimilação de dados. O primeiro experimento consiste da emulação do método BLUE quando aplicado ao modelo de Lorenz. Nesse experimento verificou-se que todos os modelos de redes neurais são capazes de emular a técnica de assimilação de dados.

Os quatro experimentos restantes consistem de variações de tamanho do grid para o modelo de Água Rasa, com o objetivo de verificar eficiência computacional. Em ambos os experimentos observou-se que o método NEAT através da busca por topologia mínima precisou de um número significativamente menor de operações para realizar um *update* quando comparado aos métodos de topologia fixa. No entanto, dada a sua complexa representação, o ganho em número de operações não se refletiu em ganho de desempenho computacional para todos os casos. Com um grid suficientemente grande, nesse casos maior que 15x15, o número de operações se refletiu em um melhor desempenho computacional.

Como trabalhos futuros, sugere-se examinar alternativas para busca de topologia mínima em redes neurais como o algoritmo Correlação Cascade (Fahlman & Lebiere, 1991), bem como a aplicação dos conhecimentos obtidos com este trabalho em um processo de assimilação de dados real como o SUPIM-DAVS.

6 ANEXO

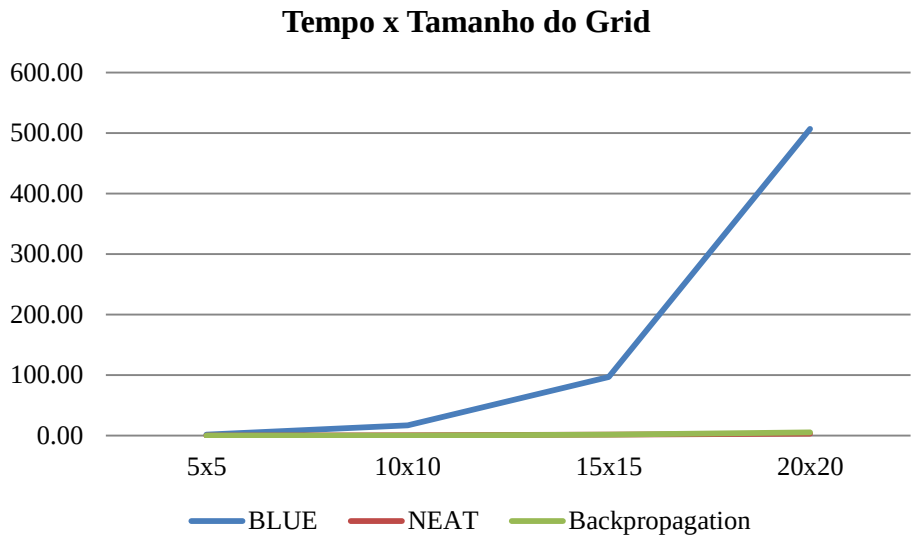

Figura 6.1: Resultados dos experimentos em relação ao tempo para efetuar assimilação de dados para todo o conjunto de previsão. Nesse caso são comparados os modelos de redes neurais com o método BLUE. Através da figura fica evidente a vantagem no uso de redes neurais na emulação de técnicas de assimilação de dados com o objetivo de ganho de desempenho.

7 REFERÊNCIAS

AForge.NET. (2011, 10 21). Retrieved 10 21, 2011, from AForge.NET: http://www.aforgenet.com/

SharpNEAT . (2011, 10 21). Retrieved 10 21, 2011, from SharpNEAT : http://sharpneat.sourceforge.net/

Bouttier, F., & Courtier, P. (1999, 04). *Data assimilation concepts and methods*. Retrieved 10 21, 2011, from European Centre for Medium-Range Weather Forecasts: http://www.ecmwf.int/newsevents/training/rcourse_notes/DATA_ASSIMILATION/ASSIM_CONCEPTS/Assim_concepts.html

Bradford, & Sanders. (2011, 10 21). *MATLAB CODES*. Retrieved 10 21, 2011, from MATLAB CODES: http://sanders.eng.uci.edu/matlabcodes.html

Bradford, S., & Sanders, B. F. (2002). Finite-Volume Models for Unidirectional, Nonlinear, Dispersive Waves. *ASCE Journal of Waterway, Port, Coastal, and Ocean Engineering*, pp. 173-182.

Buckland, M. (2002). *AI Techniques for Game Programming.* Course Technology PTR.

Cintra, R. S. (2010, setembro 27). *Assimilação de Dados com Redes Neurais Artificiais em Modelo de Circulação Geral da Atmosfera.* Tese de Doutorado do Curso de Pós-Graduação em Computação Aplicada, INPE.

Cintra, R., & Velho, H. (2010). Assimilação de Dados Utilizando Redes Neurais Artificiais em um Modelo Atmosférico de Circulação Geral. *XVI CBMET - Congresso Brasileiro de Meteorologia*.

Cintra, R., & Velho, H. (2010). Redes Neurais Artificiais na Melhoria de Desempenho de Métodos de Assimilação de Dados: Filtro de Kalman. *TEMA Tend. Mat. Apl. Comput, 11*(1), 29-39.

Cintra, R., Velho, H., & Todling, R. (2009). Assimilação de Dados Atmosféricos utilizando Redes Neurais Artificiais: Nova Abordagem. *Congresso Nacional de Redes Neurais*.

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics o Control, Signals, and Systems*, 303–314.

Daley, R. (1991). *Atmospheric Data Analysis.* Cambrige: Cambrige University Press.

Fahlman, S. E., & Lebiere, C. (1991). *The Cascade-Correlation Learning Architecture.* National Science Foundation.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley.

Hagen, T., Hjelmervik, J., Lie, J., Natvig, J., & Henriksen, M. (2005). Visual Simulation of Shallow-Water Waves. *Elsevier Programmable Graphics Hardware*, pp. 716-726.

Härter, F. (2004). *Redes Neurais Recorrentes Aplicadas a Assimilação De Dados Em Dinâmica Não-Linear.* tese de Doutorado, CAP-INPE, Computação Aplicada.

Haykin, S. (2001). *Redes neurais princípios e prática.* Porto Alegre: Bookman.

Kranenburg, C. (1993). Quasi-3D Numerical Modeling of Shallow-Water Circulation. *Journal of Hydraulic Engineering, 119*(4).

Liaqat, A., Fukuhara, M., & Takeda, T. (2001). Application of neural network collocation method to data assimilation. *Computer Physics Communications, 141*, pp. 350-364.

Lorenz, E. (1963). Deterministic non periodic flow. *Journal of theAtmosphericPhy-sics*, pp. 130–141.

Miguel, C. G. (2009). *Evolução Estrutural e Paramétrica de Redes Neurais Dinâmicas em Vida Artificial.* Universidade de São Paulo, São Paulo.

Nowosad, A. (2001). *Novas Abordagens para Assimilação de Dados Meteorológicos.* tese de Doutorado, CAP-INPE, Computação Aplicada.

Petry, A. (2002). *Reconhecimento Automático de Locutor Utilizando Medidas de Invariantes Dinâmicas Não-Lineares.* tese de Doutorado, UFRGS, Ciência da Computação.

Rojas, R. (1996). *Neural Networks: A Systematic Introduction.* Berlin: Springer-Verlag.

Rüegg, C., Cuda, M., & Gael, J. V. (2011, 10 21). *Math.NET Numerics*. Retrieved 10 21, 2011, from Math.NET Numerics: http://numerics.mathdotnet.com/

Russell, S., & Norvig, P. (2010). *Artificial Intelligence: A Modern Approach.* Prentice Hall.

Stanley, K., & Miikkulainen, R. (2002). Evolving Neural Networks through Evolving Neural Networks through. *The MIT Press Journals*, pp. 99-127.

William, W., Hsieh, & Tang, B. (1998). Applying Neural Network Models to Prediction and Data Analysis in Meteorology and Oceanography. *Bulletin of the American Meteorological Society*.

Yu, T., Iredell, M., & Keyser, D. (1997). Global Data Assimilation and Forecast Experiments Using SSM/I Wind Speed Data Derived from a Neural Network Algorithm. *Weather and Forecasting, 12*, pp. 859–865.