MINISTÉRIO DA CIÊNCIA, TECNOLOGIA, INOVAÇÕES E COMUNICAÇÕES
INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

sid.inpe.br/mtc-m21b/2017/06.26.23.06-TDI

REDE NEURAL AUTO-CONFIGURADA PARA ASSIMILAÇÃO DE DADOS USANDO FPGA PARA A CIRCULAÇÃO OCEÂNICA

Sabrina Bérgoch Monteiro Sambatti

Tese de Doutorado do Curso de Pós-Graduação em Computação Aplicada, orientada pelos Drs. Haroldo Fraga de Campos Velho, e Andrea Schwertner Charão, aprovada em 30 de maio de 2017.

URL do documento original:
<http://urlib.net/8JMKD3MGP3W34P/3P6NQ82>

INPE
São José dos Campos
2017

PUBLICADO POR:

Instituto Nacional de Pesquisas Espaciais - INPE
Gabinete do Diretor (GB)
Serviço de Informação e Documentação (SID)
Caixa Postal 515 - CEP 12.245-970
São José dos Campos - SP - Brasil
Tel.:(012) 3208-6923/6921
E-mail: pubtc@inpe.br

MINISTÉRIO DA CIÊNCIA, TECNOLOGIA, INOVAÇÕES E COMUNICAÇÕES
INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

sid.inpe.br/mtc-m21b/2017/06.26.23.06-TDI

REDE NEURAL AUTO-CONFIGURADA PARA ASSIMILAÇÃO DE DADOS USANDO FPGA PARA A CIRCULAÇÃO OCEÂNICA

Sabrina Bérgoch Monteiro Sambatti

Tese de Doutorado do Curso de Pós-Graduação em Computação Aplicada, orientada pelos Drs. Haroldo Fraga de Campos Velho, e Andrea Schwertner Charão, aprovada em 30 de maio de 2017.

URL do documento original:
<http://urlib.net/8JMKD3MGP3W34P/3P6NQ82>

INPE
São José dos Campos
2017

Dados Internacionais de Catalogação na Publicação (CIP)

Sambatti, Sabrina Bérgoch Monteiro.

Sa44r Rede neural auto-configurada para assimilação de dados usando FPGA para a circulação oceânica / Sabrina Bérgoch Monteiro Sambatti. – São José dos Campos : INPE, 2017.

xxii + 95 p. ; (sid.inpe.br/mtc-m21b/2017/06.26.23.06-TDI)

Tese (Doutorado em Computação Aplicada) – Instituto Nacional de Pesquisas Espaciais, São José dos Campos, 2017.

Orientadores : Drs. Haroldo Fraga de Campos Velho, e Andrea Schwertner Charão.

1. Filtro de Kalman. 2. Assimilação de dados. 3. rede neural artificial. 4. Computação heterogênea. 5. FPGA. I.Título.

CDU 004.032.26:551.509.3

Aluno (a): *Sabrina Bérgoch Monteiro Sambatti*

Titulo: "REDE NEURAL AUTO-CONFIGURADA PARA ASSIMILAÇÃO DE DADOS USANDO FPGA PARA A CIRCULAÇÃO OCEÂNICA"

Aprovado (a) pela Banca Examinadora em cumprimento ao requisito exigido para obtenção do Título de *Doutor(a)* em *Computação Aplicada*

Dr. Stephan Stephany

Presidente / INPE / SJCampos - SP

Dr. Haroldo Fraga de Campos Velho

Orientador(a) / INPE / São José dos Campos - SP

Dra. Andrea Schwertner Charão

Orientador(a) / UFSM / Santa Maria - RS

Dra. Rosangela Saher Correa Cintra

Convidado(a) / . / São José dos Campos - SP

Dr. Eduardo Rocha Rodrigues

Convidado(a) / IBM / Rio de Janeiro - RJ

Dr. Eduardo Marques

Convidado(a) / USP / São Carlos - SP

Este trabalho foi aprovado por:

( ) maioria simples

(x) unanimidade

São José dos Campos, 30 de maio de 2017

"Por vezes sentimos que aquilo que fazemos não é senão uma gota de água no mar. Mas o mar seria menor se lhe faltasse uma gota".

Madre Teresa de Calcuta

A Ian e Thomas

AGRADECIMENTOS

Agradeço aos meus orientadores Dr. Haroldo Fraga de Campos Velho e Dra. Andrea Schwertner Charão pela orientação, confiança e paciência.

Agradeço a todos os professores e pesquisadores da CAP.

Agradeço ao Instituto Nacional de Pesquisas Espaciais pela oportunidade e a CAPES, Coordenação de Aperfeiçoamento de Pessoal de Nível Superior, pelo apoio financeiro.

Gostaria de agradecer a Helaine Cristina Morais Furtado, Juliana Aparecida Anochi, Vitor Conrado Faria Gomes e Reynier Hernández Torres pelo suporte e apoio dado durante o desenvolvimento desta tese.

E finalmente gostaria de agradecer aos meus colegas que se tornaram meus grandes amigos!!!

RESUMO

Processos físicos podem ser matematicamente representados por equações diferenciais, mas há uma lacuna entre a representação matemática e o processo real. Com a fusão da informação observacional com os dados do modelo, o erro de simulação pode ser atenuado. Esquemas de combinação de dados observacionais com dados de um modelo de predição matemática são conhecidos como Assimilação de Dados (DA), calculando a condição inicial (análise) para um sistema dinâmico. Várias técnicas são empregadas na assimilação de dados, como: Filtro de Kalman, filtro de partículas e métodos variacionais, que são os mais pesquisados. No entanto, os métodos mencionados demandam por computação intensiva. Neste trabalho, uma Rede Neural Artificial (RNA) é projetada para emular o filtro de Kalman, com redução do esforço computacional. Geralmente, um especialista em redes neurais encontra uma arquitetura adequada após um longo trabalho empírico. Neste trabalho é adotada uma abordagem automática para identificar a melhor arquitetura para a RNA. Essa identificação é formulada como um problema de otimização, resolvido por uma nova metaheurística: Algoritmo de Colisão de Partículas Múltiplas (MPCA). A RNA ideal projetada para assimilação de dados é implementada em *software* e em FPGA (*Field-Programmable Gate Array*), um dispositivo de *hardware* usado como co-processador. O FPGA é o processador neural aplicado para a computação da condição inicial do modelo dinâmico. Dois sistemas dinâmicos são usados para testar a metodologia: a equação de onda 1D e as equações de água rasa 2D. O sistema de águas rasas está preparado para simular a circulação oceânica. Para ambos os sistemas, a RNA foi eficaz, com redução do tempo de processamento. A utilização de FPGA como co-processador para assimilação de dados tem um desempenho semelhante ao da análise calculada por *software*.

Palavras-chave: Filtro de Kalman. Assimilação de dados. Rede neural artificial. Computação heterogênea. FPGA.

SELF-CONFIGURED NEURAL NETWORK FOR DATA ASSIMILATION USING FPGA FOR OCEAN CIRCULATION

ABSTRACT

Physical processes can be mathematically represented by differential equations, but there is a gap from the real process. With the incorporation of observational information to the model state, the simulation error can be reduced. Data Assimilation (DA) is the schemes of combining observational data with data from a mathematical prediction model, computing the initial condition (analysis) for the dynamical system. Here, an Artificial Neural Network (ANN) is designed to emulate Kalman filter, with reduction of the computational effort. An automatic approach to identify the best configuration for the ANN is adopted. This issue is formulated as an optimization problem, solved by a new metaheuristic: Multiple Particles Collision Algorithm (MPCA). The optimal ANN designed for data assimilation is implemented on FPGA (Field-Programmable Gate Array), a hardware device used as a co-processor. Two dynamical systems are used to test the framework: the wave 1D equation, and the 2D shallow water equations. The shallow water system is prepared to simulate oceanic circulation. For both systems, the ANN was effective, with reduction of processing time. The use of FPGA as a co-processor for data assimilation has a similar performance than analysis calculated by software.

Keywords: Kalman filter. Data assimilation. Artificial neural network. Heterogeneous computing. FPGA.

LISTA DE FIGURAS

Pág.

LISTA DE TABELAS

Pág.

LISTA DE ABREVIATURAS E SIGLAS

INPE	–	Instituto Nacional de Pesquisas Espaciais
MPCA	–	*Multi-Particle Collision Algorithm*
PCA	–	*Particle Collision Algorithm*
MCP	–	Neurônio de *McCulloch* e *Pitts*
MLP	–	*Multilayer perceptron*
PMC	–	*Perceptron* multicamadas
ASIC	–	*Application Specific Integrated Circuits*
GPP	–	Processador de propósito geral
CMP	–	Multiprocessador de *chips* homogêneos ou heterogêneo
FPGA	–	*Field programmable gate array*
FPOA	–	*field programmable object arrays*
GPU	–	Unidade de processamento gráfico
DSP	–	Processador de sinais digitais
ASIP	–	Processador de instruções específicas de aplicações
HPEC	–	Computação embarcada de alto desempenho
HPC	–	Computação de alto desempenho
CPU	–	Unidade de processamento central
PAL	–	*Programmable Array Logic*
GAL	–	*Generic Array Logic*
CPLD	–	*Complex Programmable Logic Device*
RC	–	Computação Reconfigurável
CPU	–	Unidade central de processamento
MIC	–	*Many Integrated Cores*
DRAM	–	*Dynamic random access memory*
SRAM	–	Static Random Access Memory
RTCORE	–	API *RapidArray Transport Core*
VHSIC	–	*Very High Speed Integrated Circuits*
VHDL	–	VHSIC *Hardware Description Language*
MAC	–	*Multiplier and Accumulator*
RNA	–	Rede neural artificial
NCEP	–	*National Center for Environmental Prediction*
NOAA	–	*National Oceanic and Atmospheric Administration*
NCSA	–	*National Center for Supercomputer Applications*
MPI	–	*Message Passing Interface*
FK	–	Filtro de kalman
LAC	–	Laboratótio Associado de Computação e Matemática Aplicada
KdV	–	*Korteweg-deVries*
EQM	–	Erro quadrático médio

SUMÁRIO

Pág.

1 INTRODUÇÃO

Processos físicos podem ser descritos através de equações diferenciais utilizando modelagem matemática. Por ser uma aproximação da realidade, há erros inerentes ao processo. Um modo de compensar os erros de modelagem é adicionando informações ao modelo. Estas informações consistem de observações, isto é, dados provenientes de medidas sobre o fenômeno modelado. O procedimento que combina de maneira adequada observações e dados de um modelo matemático é conhecido como **assimilação de dados**. A metodologia é aplicada para identificar a melhor condição inicial para um modelo de previsão numérica.

O processo de assimilação ocorre da seguinte forma: cada instante que novas observações estão disponíveis, estas são usadas em conjunto com uma previsão mais recente. Estes dados combinados geram uma estimativa de estado do modelo – também chamada de análise – para a próxima integração temporal, e o procedimento se repete sistematicamente (CINTRA, 2010).

Este modo eficiente de combinar as observações com o modelo matemático podem ser baseadas em métodos da Teoria da Estimação (como o Filtro de Kalman e o Filtro de Partículas), no Cálculo Variacional, ou baseadas em técnicas de Inteligência Artificial, como o uso de Redes Neurais Artificiais (FURTADO, 2008)

O desafio atual da assimilação de dados consiste em produzir uma análise em tempo viável para a prática operacional. A previsão é realizada diariamente e deve ser executada numa janela de tempo de poucas horas. Para melhorar a previsão, é necessário aumentar a resolução do modelo, ou seja, aumentar o número de células computacionais para ter uma representação dos processos físicos de forma mais complexa e detalhada. Além disso, há um crescimento exponencial da quantidade de dados de observação disponíveis – um exemplo de excelência de *Big Data*, ou melhor: *Data Science*. Combinar de forma adequada estes conjuntos de dados requer o uso de algoritmos sofisticados e grande esforço computacional. Isso tudo representa uma grande e crescente demanda de capacidade de computação intensiva. Assimilação de dados é uma das etapas que mais demandam capacidade de processamento no processo de previsão numérica.

A abordagem para assimilação de dados por meio de Redes Neurais Artificiais vem sendo investigada como uma proposta de solução para o problema (FURTADO, 2008) e é o tema desta pesquisa de doutorado.

Redes Neurais Artificiais (RNA) é um dos temas expoentes em Inteligência Artificial. Atualmente, está sob intensa pesquisa. Embora exista muito estudo na área, há ainda muitas perguntas sobre redes neurais que precisam ser examinadas. Um dos principais tópicos em RNA é a dificuldade de se estabelecer uma arquitetura ótima, ou próximo da ótima, para o processo de configuração e treinamento de redes. O procedimento para definir uma rede neural adequada para solucionar um problema específico é uma tarefa complexa e em geral exige um grande esforço do desenvolvedor, principalmente para determinar os melhores parâmetros, e é necessário um conhecimento prévio sobre o problema a ser tratado. O processo de busca e definição de uma arquitetura ideal para uma RNA é muito relevante, exigindo uma intensa pesquisa sobre a eficiência computacional do modelo (CARVALHO, 2011b).

Existem muitos algoritmos na literatura para o treinamento da RNA, tendo como objetivo a melhoria da capacidade de generalização e a especificação de uma arquitetura ótima, tais como: (a) *Pruning* (HINTON, 1989), faz o ajuste da rede neural, modificando a sua estrutura, o treinamento começa com uma arquitetura super dimensionada, os pesos são eliminados até que a capacidade de generalização possa ser aumentada; (b) *Weight Decay* (HINTON, 1989), o algoritmo é semelhante ao *Pruning*, onde a função custo e o vetor de pesos são modificados; (c) *Early Stopping* proposto por (WEIGEND; HUBERMAN, 1990), em que a interrupção precoce do treinamento é efetuada, sem alterar a arquitetura da RNA, da mesma forma; (d) *Cross Validation* proposto por (STONE, 1978), também conhecida por Validação Cruzada, é conhecida por melhorar a generalização, onde o conjunto de dados são separados em dois conjuntos: dados de treinamento e dados de validação, o conjunto de validação é responsável por avaliar o desempenho dos modelos.

A complexidade computacional de uma arquitetura de rede neural baseado no número de neurônios e o número de épocas é proposto por Carvalho (2011b). Este procedimento é usado para definir uma RNA, e é empregado para o problema da recuperação perfis atmosféricos da concentração de gás (CARVALHO, 2011b). A função objetivo a ser resolvida por meio de técnicas de otimização é a soma da diferença quadrática entre a saída desejada e a saída da RNA e uma métrica de complexidade da rede.

Nesta tese, foi utilizado um método baseado em técnicas de otimização estocástica para identificar arquitetura ótima, ou próximo da ótima, para uma RNA. Um termo de penalidade é utilizado para avaliar a função objetivo, buscando identificar arquiteturas de rede mais simples possível. O algoritmo de otimização utilizado é o

MPCA: *Multi-Patrticle Collision Algorithm*, uma metaheurística proposta por Luz (2012). Diversos parâmetros podem ser estimados em uma RNA: número de camadas escondidas, número de neurônios artificiais, o tipo de função de ativação, taxa de aprendizado e *momentum*, que foram os parâmetros estimados neste trabalho.

Como mencionado, a metodologia de RNA foi empregada no problema de assimilação de dados, onde uma Rede Neural Perceptron de múltiplas camadas (MLP: Multilayer Perceptron) é treinada para emular o Filtro de Kalman (FK). O modelo neural de assimilação de dados é configurado de forma automática, onde a arquitetura ótima da rede é determinada pela metaheurística MPCA.

Com a demanda crescente por maior poder computacional, uma alternativa é o uso de co-processadores – tendência atual, ou *computação híbrida*. Os dispositivos disponíveis atualmente para este tipo de computação heterogênea são: GPU (Graphical Processing Unit), MIC (Multi-Integrated Core) e FPGA (Field Programmable Gate Array). Computação híbrida associando CPU com FPGA é o foco desta tese, onde o processo de assimilação de dados é realizado no co-processador. A rede neural autoconfigurada pelo MPCA que emula o filtro de Kalman para assimilação de dados é implementada em FPGA.

O processo de assimilação pelo neuro-computador em FPGA é testado inicialmente para a equação da onda unidimensional de primeira ordem (BENNETT, 2002). Esta equação é usada como um modelo padrão de equação hiperbólica. Outro sistema de teste usado foi com o modelo de água rasa (*shallow water*) 2D, que foi ajustado para descrever circulação oceânica (BENNETT, 2002). As equações de água rasa foram modeladas inicialmente pelos oceanógrafos. Uma vez que este modelo mostrou-se adequado para o estudo de propagação de ondas, o sistema passou a ser adotado também pelos meteorologistas. Pode-se dizer que as equações de água rasa são muito utilizadas para o estudo em dinâmica dos fluidos geofísicos, inclusive aplicado em investigação da execução de modelos numéricos em novas arquiteturas de computadores (FU H ; GAN, 2017). Aqui, o modelo de água rasa é usado em teste de assimilação de dados em computação heterogênea.

1.1 Organização da tese

A tese se organiza em capítulos, como segue:

- Capítulo 2: Conceitos de assimilação de dados e descrição do filtro de Kalman.

- Capítulo 3: Apresentação de rede MLP, estratégia de configuração automática e metaheurística MPCA.

- Capítulo 4: Computação com FPGA e o computador de processamento heterogêneo Cray XD1.

- Capítulo 5: Os modelos usados nos testes são detalhados: equação de Onda 1D de primeira ordem e sistema de água rasa 2D.

- Capítulo 6: Apresenta resultados e comentários.

- Capítulo 7: Mostra as conclusões da tese e são feitos comentários de trabalhos futuros.

2 ASSIMILAÇÃO DE DADOS

Assimilação de dados é um processo usado para identificar a melhor condição inicial para um sistema de previsão, o qual é baseado em integração temporal de um modelo de simulação baseado em equações diferenciais ou integro-diferenciais. A estimativa da melhor condição inicial está baseada num processo de fusão de dados de observação com dados do modelo de simulação. O resultado do processo de assimilação de dados é chamado de *análise*, que representa a condição inicial a ser usada pelo modelo de previsão.

A análise é uma imagem precisa do verdadeiro estado da atmosfera em um determinado momento, sendo representado em um modelo como uma coleção de números. A análise é útil pois representa uma descrição ampla e consistente da atmosfera. Pode ser usada como dados de entrada para outra operação, em particular como condição inicial para uma previsão numérica do tempo, e também pode fornecer uma referência para verificar a qualidade das observações (BOUTTIER; COURTIER, 2002).

Do ponto de vista científico, a primeira análise explícita do problema de previsão do tempo foi realizada no início do século XX, quando o cientista norueguês Vilhelm Bjerknes desenvolveu a teoria das frentes frias e quentes, e estabeleceu um plano de duas etapas para a previsão: diagnóstico e prognóstico. Na etapa de diagnóstico ele utilizou dados observacionais adequados para definir a estrutura tridimensional da atmosfera em um determinado momento. Já o prognóstico, segunda etapa, foi provido pela montagem de um conjunto de equações, uma para cada variável dependente descrevendo a atmosfera (RICHARDSON, 2007).

O meteorologista e matemático Lewis Fry Richardson propôs o primeiro método matemático sistêmico para prever o tempo e demonstrou sua aplicação realizando uma previsão de ensaio. O método de previsão de Richardson equivale a uma implementação precisa e detalhada da etapa de prognóstico de Bjerknes. Contudo, por envolver um enorme volume de computação numérica tornou-se impraticável na era pré-computador digital (LYNCH, 2006).

Através da evolução dos computadores, dos modelos numéricos e da rede de observação meteorológica, a previsão numérica passou a ter melhor qualidade. Porém, todo e qualquer sistema de previsão, que se baseia na metodologia inaugurada pelos meteorologistas, necessita de uma adesão com a observação para evitar que o resultado do modelo se distancie da realidade. Como todos os modelos dinâmicos fundamen-

tados em conceitos matemáticos constituem apenas aproximações da realidade, é necessário alguma metodologia para tornar a previsão o mais fiel possível à realidade. Os modelos de previsão são aplicadas em várias áreas: dinâmica da atmosfera, oceano, vazão de rios, dinâmica da ionosfera ou evolução de um câncer.

A assimilação de dados e previsão percorre o seguinte ciclo: a previsão se desloca da realidade a cada passo de tempo, tornando-se necessário inserir informações do sistema observado, que irá gerar uma nova análise - a condição inicial. No entanto, mesmo considerando, por hipótese, que os modelos sejam perfeitos, medidas de observação contém erros e em sistemas caóticos, qualquer pequena alteração na condição inicial alterará a dinâmica, tornando a assimilação de dados um processo necessário e de extrema importância.

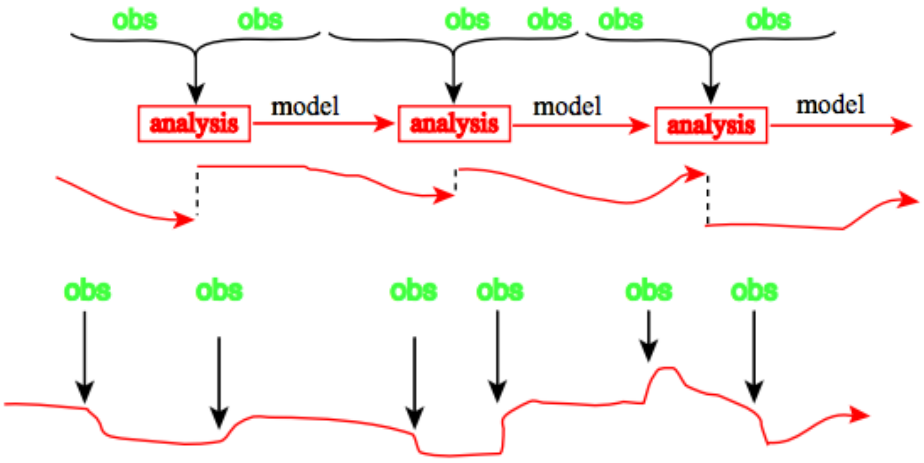
O problema da determinação das condições iniciais para um modelo de previsão é muito importante e complexo, e tornou-se uma ciência em si. Como a integração temporal de um modelo de previsão é um problema de valor inicial, a capacidade de fazer uma previsão de qualidade requer que o modelo computacional seja uma representação realista do fenômeno físico e que as condições iniciais sejam as mais precisas possíveis, evidentemente quanto mais precisa a estimativa das condições iniciais, melhor a qualidade das previsões.

A melhoria na habilidade de previsão numérica do tempo nos últimos 40 anos deve-se a alguns fatores, dentre eles, o uso de métodos mais precisos de assimilação de dados resultando em melhores condições iniciais para os modelos. Atualmente, os centros operacionais de previsão geram condições iniciais através de uma combinação de observações e previsões de curto alcance. Essa metodologia, conhecida como assimilação de dados, cujo propósito foi definido por Talagrand e Courtier (1987) é exemplificado pela seguinte ideia – "usar toda a informação disponível, para determinar o mais precisamente possível o estado do fluxo atmosférico (ou oceânico)" (KALNAY, 2003). De modo simplificado, pode-se dizer que a assimilação de dados é a ciência de ter uma combinação adequada de dados de um modelo matemático de previsão com dados de observação do sistema em estudo (FURTADO, 2008).

A assimilação de dados é uma técnica de análise em que a informação observada é acumulada no estado do modelo, aproveitando as restrições de consistência com leis de evolução temporal e propriedades físicas. Existem duas abordagens básicas para a assimilação de dados segundo Bouttier e Courtier (2002):

- assimilação sequencial: que só considera a observação feita no passado até

Figura 2.1 - Representação de duas estratégias básicas de assimilação de dados – sequencial intermitente e sequencial contínua, em função do tempo



Fonte: (BOUTTIER; COURTIER, 2002)

o momento da análise, que é o caso dos sistemas de assimilação em tempo real;

- assimilação não-sequencial ou retrospectiva: a observação do futuro pode ser usado, por exemplo, em um exercício de reanálise.

Existe outra distinção que pode ser utilizada entre os métodos de assimilação que é: intermitente ou contínua no tempo. No método intermitente, as observações podem ser processadas em pequenos lotes, o que é tecnicamente conveniente. E no método contínuo, os lotes de observação durante períodos mais longos são considerados, e a correção para o estado analisado é suave no tempo, que é fisicamente mais realista. Uma descrição esquemática de duas abordagens é mostrada na Figura 2.1: assimilação sequencial intermitente e assimilação sequencial contínua, destaca-se a forma como a distribuição temporal das observações ("obs") é processada para produzir uma sequência temporal de estados assimilados (a curva inferior em cada figura) pode ser sequencial e/ou contínua (BOUTTIER; COURTIER, 2002).

Uma diferença importante entre a assimilação de dados em oceanografia e meteorologia está na motivação. Assimilação atmosférica foi impulsionada pela necessidade de prever, assim sendo, a assimilação oceanográfica, no presente e no futuro, é e será impulsionada pela necessidade de compreender melhor a dinâmica oceânica através da combinação de dados observados com dados do modelo das mesmas variáveis

dinâmicas (GHIL; MALANOTTE-RIZZOLI, 1991).

Quanto a natureza da propriedade estimada, o processo de assimilação de dados é considerado um problema inverso de determinação de condição inicial (VELHO, 2005).

Quanto mais exato for a estimativa da condição inicial, melhor será a qualidade da previsão. Para isso, necessita-se da utilização de ferramentas de assimilação de dados para inicializar os modelos numéricos de previsão. Matematicamente a assimilação de dados é um processo de dois passos:

(i) Passo da previsão:

$$x_i^p = F(x_{i-1}^a) \tag{2.1}$$

(ii) Passo de análise:

$$x_i^a = x_i^p + \rho \tag{2.2}$$

em que x_i^p é o vetor de variáveis de estado prevista do modelo, os sobrescritos p e a representam os passos de previsão e análise, respectivamente. F representa o modelo numérico, ρ é o incremento de análise ou inovação, que é determinado de acordo com a técnica da assimilação utilizada, x_i^a representa o dado de análise ou condição inicial (c.i.) e índice i denota o tempo discreto.

Os métodos de assimilação de dados podem ser classificados em (HÄRTER, 2004):

- determinístico: transformada de Laplace, relaxação Newtoniana – *Nudging*
- estocástico: métodos variacionais, filtro de Kalman

A evolução histórica dos métodos de assimilação de dados passa pela relaxação Newtoniana (*Nudging*), correções sucessivas, interpolação ótima, métodos variacionais e filtro de Kalman. O filtro de Kalman é o método estocástico que mais se destaca, além desses existe, também, a rede neural artificial que pode ser usada como um método de assimilação, e que representa uma das pesquisas desenvolvidas no Laboratório Associado de Computação e Matemática Aplicada (LAC) do INPE, que foi pioneiro na aplicação de redes neurais para assimilação de dados (NOWOSAD, 2001). Nesta tese foi usada uma rede neural Perceptron de múltiplas camadas treinada para

emular o filtro de Kalman.

2.1 Filtro de Kalman

O filtro de Kalman é uma técnica de estimação derivada da técnica dos mínimos quadrados recursivos, que leva em consideração o modelo estatístico do tipo gaussiano. Para a completa descrição de um modelo gaussiano, basta conhecer a média e o desvio padrão. Num processo estocástico, a distribuição, ou ainda, as propriedades estatísticas, vão variar com o tempo. Assim, se a variação no tempo da distribuição (Função de Distribuição e Probabilidade) for conhecida, o processo estocástico ficará bem determinado.

A melhor estimativa do estado da atmosfera (análise) é obtida, tal como indicado por Talagrand e Courtier (1987) ao combinar informações prévias sobre a atmosfera (*background* ou primeira hipótese) com observações, mas para realizar essa combinação de forma ótima é necessário também dos dados do erro de modelagem e da observação (KALNAY, 2003).

Johann Carl Friedrich Gauss inventou e usou o método de mínimos quadrados como técnica de estimação, no qual o valor mais provável das quantidades desconhecidas será aquele em que a soma dos quadrados das diferenças entre os valores observados e os calculados multiplicados por números que medem o grau de precisão é um mínimo, conhecido por residual. (SORENSON, 1970)

A designação do termo de mínimos quadrados está relacionado diretamente com a ideia do quadrado do erro. Esta metodologia é uma técnica de otimização que tem por objetivo determinar o melhor ajuste para um conjunto de dados. Dado o sistema $Fx = z$, caso não seja um problema bem-posto (HADAMARD, 1952), o melhor estado – x – em um conjunto com as observações z é aquele que irá minimizar a diferença quadrática $||Fx - z||^2$, onde F é o operador que transforma z em x (STRANG, 1986).

Para tal, considera-se o seguinte modelo linear e o erro de observação ϵ:

$$z = Fx + \epsilon \tag{2.3}$$

$$\epsilon = z - Fx \tag{2.4}$$

A solução deste sistema consiste em minimizar o erro ϵ com o intuito de obter a melhor aproximação, estimativa ($\widehat{x}$) do estado de z, baseado no critério quadrático,

obtido por meio da minimização do seguinte funcional:

$$J(x) = \epsilon^T \epsilon = ||\epsilon^2|| \tag{2.5}$$

em que ϵ^T é dado pela Equação 2.4, obtem-se:

$$J(x) = x^T F^T F x - x^T F^T z - z^T F x + z^T z \tag{2.6}$$

sendo que o resultado da derivada primeira de $J(x)$ em relação a x resulta na seguinte expressão:

$$\widehat{x} = (F^T F)^{-1} F^T z \tag{2.7}$$

ou seja, o vetor x que minimiza $||Fx - z||_2^2$ é obtida com a solução da equação normal $F^T F x = F^T z$, sendo o vetor $\widehat{x} = (F^T F)^{-1} F^T z$ a solução por mínimos quadrados para $Fx = z$ (HÄRTER, 2004).

Porém, se as observações tiverem diferentes valores de confiabilidade, ou seja, procedente de uma amostra maior ou uma medida mais cuidadosa, então a minimização será:

$$\theta_1^2 (Fx - z)_1^2 + \theta_2^2 (Fx - z)_2^2 + \cdots + \theta_i^2 (Fx - z)_i^2 \tag{2.8}$$

se a primeira medida for mais confiável que a segunda, então $\theta_1 > \theta_2$, portanto as observações são ponderadas de acordo com a confiabilidade nelas depositadas, ou seja, busca-se a melhor solução para os mínimos quadrados por $\theta F x = \theta z$. Desse modo, o primeiro passo é determinar o melhor x para uma dada matriz θ.

$$\widehat{x} = (F^T C F)^{-1} F^T C z \tag{2.9}$$

em que $C = \theta^T \theta$.

As propriedades estatísticas do experimento são dadas por θ, e para calcula-las, assume-se que o ruído de observação $\epsilon = z - Fx$ é não tendencioso. Na prática os ruídos são independentes, portanto a covariância é nula, fora da diagonal principal, e os pesos θ são dados por $\theta_1 = 1/\rho_i$, ou seja, quanto menor for a variância ρ_1 mais confiáveis serão as observações e maiores serão os valores do pesos θ. Observa-se que θ e $C = \theta^T \theta$, são diagonais, C contém os números $1/\rho_i$ e no caso em que as variâncias forem todas iguais, tem-se os mínimos quadrados. Como C torna o estimador por mínimos quadrados ponderados o "melhor estimador", desse modo define-se uma matriz de covariância dos erros de observação R, tal que $C = R^{-1}$. A matriz de

covariância R pode ser escrita como

$$R = E[\epsilon\epsilon^T] \ . \tag{2.10}$$

Caso o modelo trabalha com medições que chegam a cada instante, cada novo dado representa uma alteração na estimação, como o foco está somente na mudança que ocorre em x, será usado o método dos mínimos quadrados recursivos, pretende-se estimar x_1 com base na estimativa x_0, agregando a nova observação z_1.

No cálculo de x_1, utiliza-se a mesma suposição feita para calcular x_0, isto é, a matriz C que torna $\widehat{x}_1$ a melhor estimativa é $C = R^{-1}$, em que R é a matriz de covariância de observação, sendo que a expressão para covariância do erro para os mínimos quadrados recursivos é dada a seguir:

$$P_i^{-1} = P_{i-1}^{-1} F_i^T R_i^{-1} F_i \tag{2.11}$$

Com relação a expressão de atualização por mínimos quadrados recursivos para $\widehat{x}$ dadas z_i medidas, tem-se

$$x_i = x_{i-1} + G_i(z_i - F_i x_{i-1}) \tag{2.12}$$

em que $G_i = P_i F_i^T R_i^{-1}$ corresponde a matriz ganho, $z_i - F_i x_{i-1}$ corresponde a inovação e $G_i(z_i - F_i x_{i-1})$ refere-se a correção (HÄRTER, 2004).

A partir das expressões da covariância do erro (2.11) e da atualização por mínimos quadrados recursivos (2.12) pode-se derivar as equações relacionadas a aplicação do filtro de Kalman.

Considerando o seguinte sistema dinâmico:

$$\begin{aligned} z_i &= F_i x_i + \epsilon_i \\ x_{i+1} &= F_i x_i + \eta_i \end{aligned} \tag{2.13}$$

no qual o ruído no sistema dinâmico (η_i) e o ruido nas medidas (ϵ_i) são vetores aleatórios gaussianos, o subíndice i refere-se ao i-ésimo instante no tempo discreto. Calcula-se a melhor estimativa para x_0 e x_1, como também para x_2, apoiado em todas as informações disponíveis até o tempo $i = 2$. Estas estimativas são escritas: $x_{0|2}$, $x_{0|1}$ e $x_{2|2}$. E grande parte das aplicações concentra-se no novo valor de $x_{2|2}$ que

prediz $x_{3|2} = Fx_{2|2}$ no instante seguinte. Portanto z_3 corrigirá o valor predito para um valor filtrado, depois de um passo a mais. Este é apenas um caso não provável que z_3 corresponderá exatamente a predição $F_3 x_{3|2}$. A inovação, que é a diferença entre os dois, será zero, pois $x_{3|3}$ corresponderá ao valor predito $x_{3|2}$. O problema é uma extensão direta dos mínimos quadrados recursivos, no qual continuamente estima-se o mesmo vetor x (STRANG, 1986).

No filtro de Kalman não é razoável supor que $x_{i+1} = F_i x_i$ é exato, dado que o modelo é sempre imperfeito. Os erros ϵ_i e η_i não possuem a mesma dimensão, pois não tem a mesma medida e a mesma unidade. Frequentemente os erros ϵ_i são independentes e com variância ρ, os erros η_i também são independentes (FURTADO, 2011).

Na solução proposta por Kalman é calculado o melhor estimador linear não tendencioso no tempo i. Há dois componentes no processo: atualização e propagação. No passo de atualização, determina-se a estimativa $\widehat{x}$ no instante i dadas as medidas z_i. No passo de propagação, calcula-se a estimativa $\widehat{x}$ no instante $i+1$, dadas as medidas z_i. Os passos do algoritmo é apresentado a seguir:

a) Previsão baseado no modelo e cálculo da matriz de covariância da previsão

$$x_{i+1}^p = F_{i+1} x_i^a \tag{2.14}$$
$$P_{i+1}^p = F_i P_i^a F_i^T + Q_i \tag{2.15}$$

b) Cálculo do ganho de Kalman

$$G_{i+1} = P_{i+1}^p F_{i+1}^T [R_{i+1} + F_{i+1} P_{i+1}^p F_{i+1}^T]^{-1} \tag{2.16}$$

c) Cálculo da estimativa

$$z_{i+1}^p = F_{i+1} x_{i+1}^p x_{i+1}^a = x_{i+1}^p + G_{i+1}(z_{i+1} - z_{i+1}^p) \tag{2.17}$$

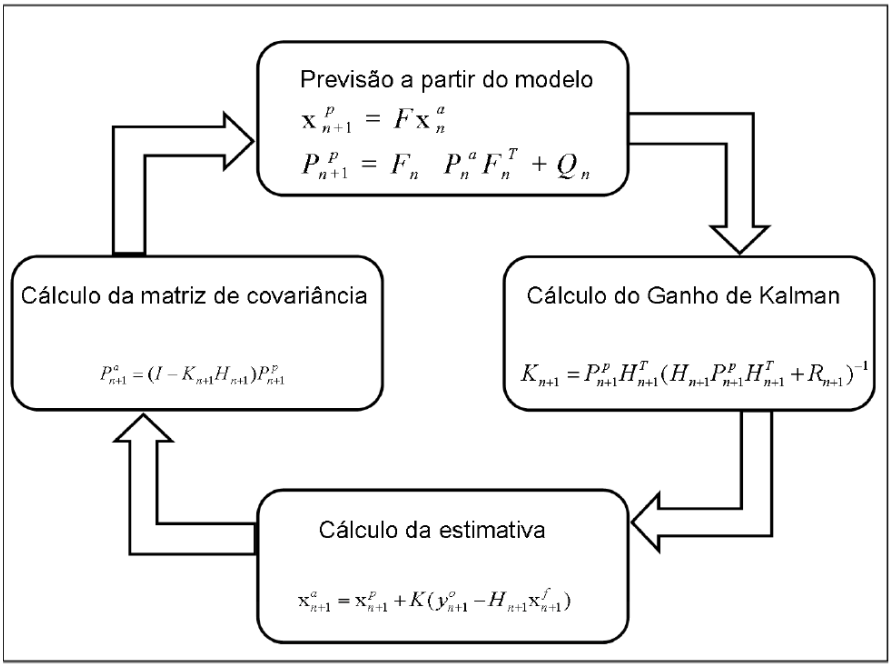
d) Cálculo da matriz de covariância da análise

$$P_{i+1}^a = [I - G_{i+1} F_{i+1}] P_{i+1}^p \tag{2.18}$$

O sobre-índice p corresponde ao passo de propagação refere-se a previsão, e o sobre-índice a refere-se ao passo de atualização e corresponde ao dado de análise no pro-

cesso de assimilação de dados. Para uma aplicação do filtro de Kalman, assume-se que os resíduos e ruídos $(z_i - F_i x_i)$ são independentes. A distribuição de densidade e probabilidade do ruído e das variáveis de estado inicial do modelo possuem distribuição gaussiana. Em cada passo de tempo as variáveis de estado e as observações manterão a distribuição gaussiana devido a linearidade do sistema (FURTADO, 2012). O algoritmo do filtro de Kalman no contexto de assimilação de dados é mostrado no esquema apresentado na Figura 2.2

Figura 2.2 - Diagrama esquematico do Filtro de Kalman Linear



Fonte: Todman et al. (2005)

O fato do filtro de Kalman ser uma solução genérica e tratar o ruído aleatório o difere dos mínimos quadrados ponderados recursivos, que foi desenvolvido apenas para parâmetros. Porém uma desvantagem do filtro de Kalman para assimilação de dados é o seu alto custo computacional, pois em todo o algoritmo existem operações matriciais computacionalmente pesadas como o processo de inversão de matrizes de grande escala. Uma forma de reduzir este custo é através do uso de redes neurais artificiais treinada para emular o filtro de Kalman, em Cintra e Campos Velho (2011) é mostrado que o desempenho computacional da rede neural artificial foi superior ao desempenho do sistema com filtro de Kalman, os resultados mostram que a eficiência

computacional da rede neural para o problema da assimilação de dados atmosféricos e faz uso de 75% menos tempo de execução e a mesma qualidade no campo de análise resultante.

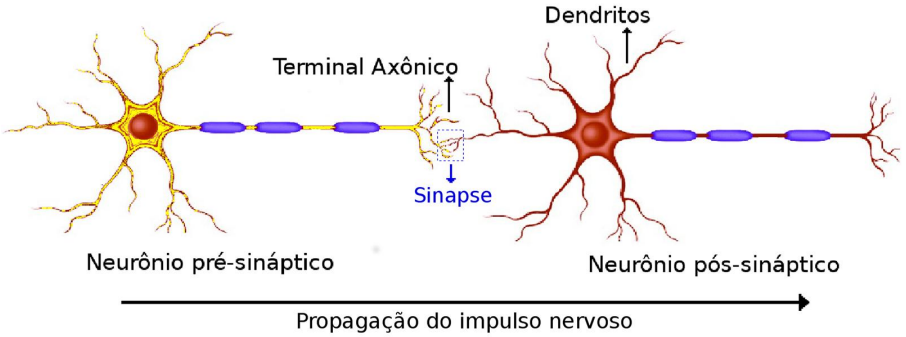
3 REDES NEURAIS

Antes mesmo do computador se tornar popular, a simulação do comportamento do cérebro humano por parte das máquinas já era ambicionada. Allan Turing, conhecido como o pai da computação, foi um dos primeiros a divulgar essa ideia através do seu clássico artigo *"Computing machinery and intelligence"*, no qual é proposta uma forma para saber se uma máquina pode ser considerada inteligente, ou seja, se pode ou não ocupar o lugar de um ser humano em um diálogo (TURING, 1950).

O que sustenta essa ideia é o fato de que o cérebro humano é um potente computador, sendo capaz de realizar certos processamentos de forma quase instantânea, como por exemplo a visão humana e o reconhecimento de padrões (HAYKIN, 2001). No entanto, em virtude da sua complexa estrutura, o cérebro ainda não foi totalmente desvendado, mas o que se sabe sobre ele é suficiente para que atualmente tenha-se poderosas ferramentas bio-inspiradas.

O cérebro é composto por uma rede de neurônios biológicos – sua célula fundamental – interligados. Cada neurônio, responsável pela recepção e transmissão de impulsos nervosos, possui três seções: corpo celular (soma), dendritos e axônio. Os dendritos recebem as descargas elétricas provenientes de outros neurônios e as transmitem para o corpo celular – ou centro metabólico do neurônio. O axônio conduz esses impulsos elétricos do corpo celular até outros locais mais distantes, através da diferença de potencial ocasionada pela bomba de sódio e potássio. Os impulsos nervosos passam de um neurônio para outro pela sinapse, local de grande proximidade entre o axônio e um dendrito. Na Figura 3.1 é possível observar um neurônio biológico pré e pós-sináptico.

Figura 3.1 - Neurônio biológico

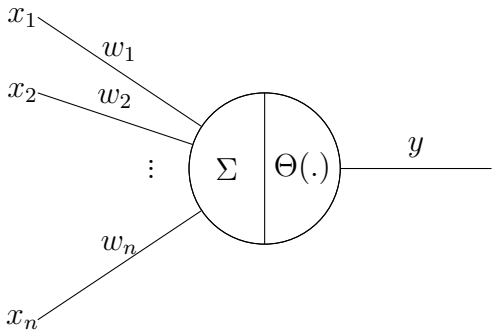


Fonte: (BORGES et al., 2015)

Em 1943 surgiram as primeiras informações sobre neurocomputação, o psiquiatra e neurologista Warren S. McCulloch em conjunto com o matemático Walter Pitts mostraram que as operações de uma célula nervosa e suas conexões podiam ser modeladas aplicando a lógica matemática. Percebeu-se claramente que o funcionamento do cérebro humano com os impulsos elétricos era baseado nas propriedades da lei "tudo ou nada" (LUCAS, 1909), pois o neurônio sempre assumiria um entre dois estados: ativado ou desativado – circuito binário – gerando o primeiro modelo artificial de um neurônio biológico (MCCULLOCH; PITTS, 1943).

McCulloch e Pitts dedicaram-se a descrever as capacidades computacionais de um neurônio artificial, resultando num modelo com n entradas – dendritos – e apenas uma saída – axônio. As sinapses são representadas pelos pesos conectados as entradas, e o neurônio é ativado através do emprego de uma função de ativação, responsável por gerar a saída do neurônio com base nos valores dos pesos e das entradas associadas (BRAGA et al., 2000). A Figura 3.2 explicita o modelo do neurônio de McCulloch e Pitts – MCP, pode-se verificar a presença dos componentes principais do modelo: entradas, pesos, função de ativação e saída.

Figura 3.2 - Neurônio artificial, em que $\sum$ corresponde a soma ponderada das entradas e a função de ativação é representada por $\Theta(.)$



Fonte: (BRAGA et al., 2000)

As técnicas de aprendizado não foram o foco do trabalho de McCulloch e Pitts, esse assunto foi abordado somente em 1949 por Donald O. Hebb, um biólogo que estudava o comportamento dos animais, em seu trabalho intitulado *"The organization of behavior: A neuropsychological theory"* (HEBB, 1949). Hebb procurou compreender como o funcionamento fisiológico dos neurônios poderiam contribuir no processo de aprendizagem, pois com a variação dos pesos era possível atingir a plasticidade

cerebral, que consiste na capacidade de rearranjo por parte das redes neuronais reformulando as suas conexões em função das necessidades, tornando-se o primeiro a propôr uma regra de aprendizagem para as sinapses dos neurônios, a regra de Hebb (HEBB, 1949):

"Quando um axônio da célula A é próximo o suficiente de excitar uma célula B e repetidamente ou persistentemente toma parte em dispará-la, algum processo de crescimento ou mudança metabólica acontece em uma ou ambas as células tal que a eficiência de A, como uma das células que disparam B, é aumentada".

A evolução do neurônio artificial passou por vários estágios. Mas somente em 1958 foi implementado por Frank Rosenblatt (ROSENBLATT, 1958) o perceptron, primeiro modelo de rede neural artificial (RNA) para aprendizagem supervisionada, é formado por neurônios MCP, possui uma topologia simples com uma camada de entrada e uma camada de saída, um peso conectado a cada entrada, sendo capaz de classificar padrões linearmente separáveis.

Depois de décadas de inércia em redes neurais dois artigos importantes fizeram com que o interesse nessa área ressurgisse. O primeiro foi a rede recorrente de John Joseph Hopfield e as propriedades associativas das RNAs (HOPFIELD, 1982). Anos mais tarde o algoritmo de aprendizagem *backpropagation* – retropopagação do erro – foi proposto por David Rumelhart, Geoffrey E. Hinton e Ronald J. Williams (RUMELHART et al., 1986), no qual o algoritmo ajusta repetidamente os pesos das ligações na rede de modo a minimizar uma medida da diferença entre o vetor de saída real da rede e a resposta desejada.

Sintetizando, rede neural artificial é uma máquina projetada para modelar o funcionamento do cérebro humano ao realizar tarefas específicas. Interligações maciças de células computacionais simples, denominadas neurônios ou unidades de processamento, são empregadas. A solução de problemas através de RNAs é bastante atrativa devido a forma com que são representados internamente pela rede, e o paralelismo inerente à arquitetura criam possibilidades de um desempenho melhor (BRAGA et al., 2000).

Seguindo a definição de Haykin (2001), a RNA se assemelha ao cérebro humano em dois aspectos:

(i) O conhecimento é adquirido pela rede através de um processo de aprendizagem;

(ii) Pesos sinápticos – forças de conexão entre neurônios – são utilizados para armazenar o conhecimento adquirido.

Outra boa definição de RNAs é conferida Braga et al. (2000), na qual afirma que são sistemas paralelos distribuídos, compostos por nós – unidades de processamento simples – responsáveis pelo cálculo de funções matemáticas (normalmente não-lineares). Estas unidades são dispostas em uma ou mais camadas interligadas por um grande número de conexões, que normalmente estão associadas a pesos, responsáveis pelo armazenamento do conhecimento representado no modelo, servindo também para equacionar a entrada recebida por cada neurônio da rede.

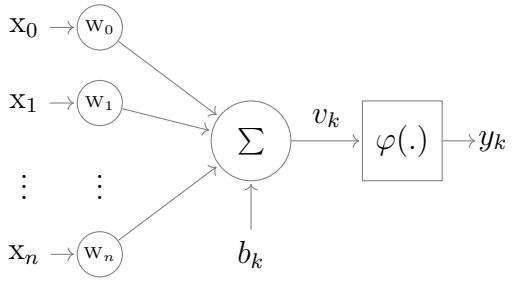
Em termos matemáticos, um neurônio k pode ser descrito considerando as seguintes equações (HAYKIN, 2001):

$$v_k = \sum_{j=1}^{n} w_{kj}\, x_j \tag{3.1}$$

$$y_k = \varphi(v_k + b_k) \tag{3.2}$$

em que x_j para $j = 1, 2, ..., n$ são os sinais de entrada; w_{kj} são os pesos sinápticos do neurônio k com a entrada j; v_k é a combinação linear entre os sinais de entrada; b_k é o viés(*bias*); φ é a função de ativação; e y_k é o sinal de saída do neurônio. O uso do viés – b_k – tem o efeito de aplicar uma transformação na saída do combinador linear – v_k – no modelo. A Figura 3.3 mostra uma representação esquemática de um neurônio artificial e seus elementos principais.

Figura 3.3 - Neurônio Artificial

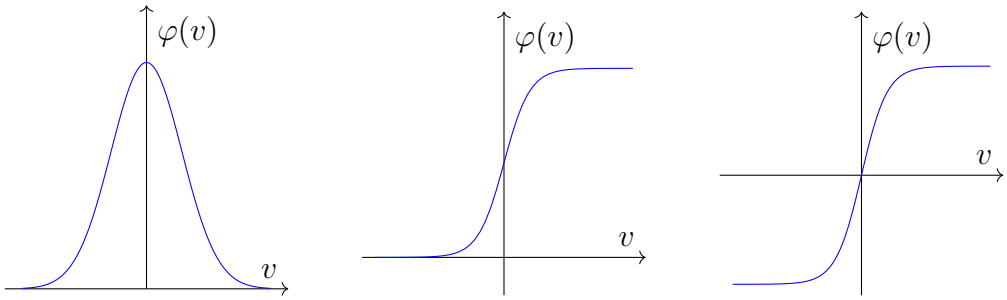


Fonte: (HAYKIN, 2001)

A escolha da função de ativação, representada por $\varphi(.)$, define a saída do neurônio

em termos do campo local induzido – v – e tem forte correlação com o problema tratado, portanto a definição de qual tipo usar é de extrema importância. Existem diversas funções de ativação que podem ser utilizadas para criar neurônios distintos, sendo que as mais usadas são (HAYKIN, 2001): (a) gaussiana, (b) sigmoide e (c) tangente, mostradas na Figura 3.4:

Figura 3.4 - Função de ativação



(a) $\varphi(v) = e^{-av}$ (b) $\varphi(v) = \frac{1}{1+e^{-av}}$ (c) $\varphi(v) = \frac{1-e^{-av}}{1+e^{-av}}$

Assim como no cérebro humano, a capacidade plástica da RNA é um mecanismo que possibilita que as sinapses se tornem mais fortes ou mais fracas dependendo da atividade dos neurônios pré e pós-sinápticos, a partir da experiência e do comportamento do indivíduo com o meio.

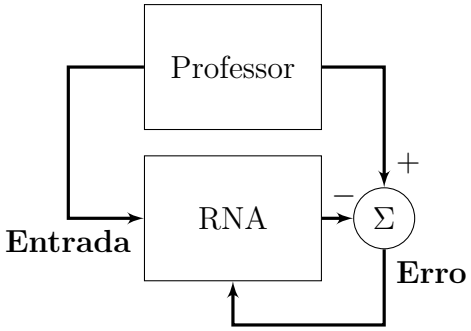
O funcionamento de uma rede neural é realizado através de duas fases: a fase da aprendizagem ou treinamento, e a fase da ativação. No processo de aprendizagem um conjunto de exemplos são apresentados à rede, que extrai as características necessárias para representar a informação fornecida. Essas características são armazenadas nos pesos sinápticos e viés, utilizados posteriormente no processo de ativação da rede, gerando respostas para o problema (BRAGA et al., 2000).

A etapa de aprendizagem de uma RNA consiste em um processo iterativo de ajuste de parâmetros. No contexto de rede neural, este processo pode ser definido como um conjunto de regras bem determinadas para solucionar um problema específico. O principal atrativo em RNAs é a habilidade de aprender através de exemplos e de generalizar a informação aprendida (BRAGA et al., 2000).

Os algoritmos de treinamento mais utilizados podem ser divididos em duas clas-

ses: aprendizado supervisionado e aprendizado não supervisionado. No aprendizado supervisionado, padrões de entrada e saída desejada são fornecidos por um supervisor externo para definir os parâmetros da rede, com o objetivo de encontrar um peso sináptico ideal entre os pares de entrada e saída fornecidos. Para este tipo de treinamento, a saída é comparada com a resposta desejada e os pesos sinápticos são ajustados com o intuito de minimizar o erro (BRAGA et al., 2000). A Figura 3.5 corresponde a uma visão esquemática do aprendizado supervisionado.

Figura 3.5 - Aprendizado supervisionado



No aprendizado não supervisionado, como o próprio nome sugere, não existe um supervisor para indicar qual a saída desejada para o padrão de entrada. Somente os padrões de entrada são apresentados para a rede. Este tipo de aprendizado, só se torna possível, quando existe redundância nos dados de entrada.

A definição de como um conjunto de neurônios artificiais estão ligados e o direcionamento das conexões sinápticas é o que se chama de arquitetura de um rede neural artificial, independente da função de ativação selecionada. A arquitetura deve ser escolhida de acordo com o problema que será abordado com base em alguns fatores como: complexidade do problema, dimensionalidade do espaço de entrada, características dinâmicas ou estáticas; conhecimento *a priori* sobre o problema; representatividade dos dados (BRAGA et al., 2000).

Segundo Haykin (2001), em geral, é possível identificar três classes de arquiteturas de RNAs fundamentalmente distintas:

(i) Redes de camada única;

(ii) Redes de múltiplas camadas;

(iii) Redes recorrentes.

A rede de múltiplas camadas é uma das mais utilizadas, nela os neurônios são organizados em uma camada de entrada, uma camada de saída e pelo menos uma camada oculta. As entradas são conectadas aos elementos processadores básicos – neurônios – que por sua vez, são interconectados aos elementos das outras camadas.

3.1 Rede Perceptron de Múltiplas Camadas

Rosenblatt (1958) apresentou o perceptron de camada única como a arquitetura mais simples de rede neural, capaz de resolver apenas problemas com características lineares, que basicamente consiste num único neurônio, pesos sinápticos e viés ajustáveis. Contudo, a não linearidade é característica predominante de situações reais, tornando-se necessário a utilização de redes com propriedades não lineares para a solução de problemas mais complexos (BRAGA et al., 2000). Surgiu então o perceptron de múltiplas camadas (PMC) – *multilayer perceptron (MLP)* – que é uma generalização do perceptron camada única, mas com poder computacional muito superior a perceptron de Rosenblatt (1958).

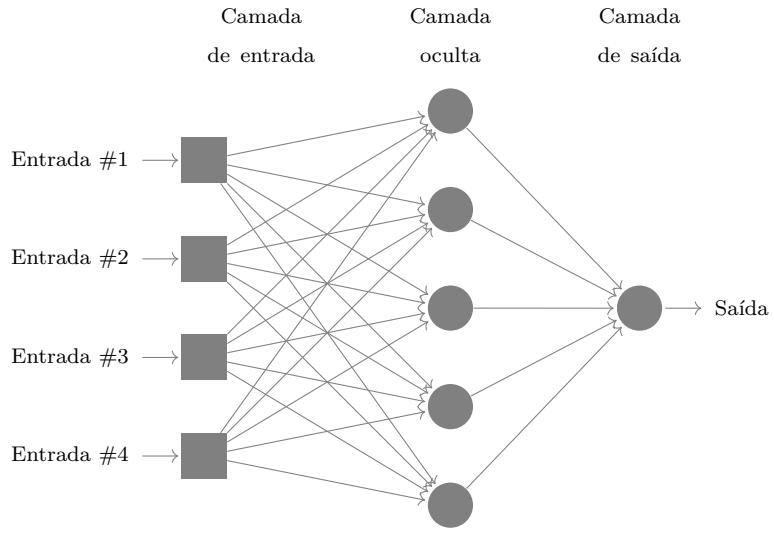
Perceptrons de múltiplas camadas são redes neurais que apresentam: camada de entrada, na qual os padrões são apresentados para a rede, pelo menos uma cada intermediária ou oculta, que trabalha como um reconhecedor de características que são armazenados nos pesos sinápticos e a camada de saída, em que os resultados são apresentados. Geralmente utilizam treinamento supervisionado em conjunto com o algoritmo de retropropagação do erro – *backpropagation* – uma generalização do algoritmo proposto por Widrow et al. (1960) – *Least Mean Square Error*, também conhecido como Regra Delta. A Figura 3.6 mostra uma arquitetura padrão de uma rede perceptron múltiplas camadas totalmente conectada com apenas uma camada oculta e uma camada de saída.

Nesta rede, a camada de entrada é singular, uma vez que não realiza qualquer processamento, sua tarefa se restringe apenas a enviar os valores de entrada para todos os neurônios da camada seguinte. A função das múltiplas camadas é transformar o problema descrito pelo conjunto de dados de entrada de tal forma que possibilite que a camada de saída possa resolver o problema retratado (BRAGA et al., 2000).

O desenvolvimento do algoritmo de retropropagação do erro representou um marco na área de redes neurais, visto que oferece um método eficiente para o treinamento de perceptrons de múltiplas camadas (HAYKIN, 2001). No algoritmo de retropropagação

do erro é utilizado pares de entrada e saída desejada (x e d) para ajustar os pesos através de um mecanismo de correção. Neste caso, o treinamento ocorre em dois passos através das diferentes camadas da rede: o passo para frente e o passo para trás.

Figura 3.6 - Perceptron de múltiplas camadas



No etapa de propagação (*forward*), um conjunto de padrões é apresentado à camada de entrada e o vetor resposta desejada é apresentado à camada de saída. O campo local induzido – soma ponderada de todas as entradas sinápticas acrescidas do bias – de um determinado neurônio j é representado por $v_j(n)$ e definido na Equação 3.3:

$$v_j(n) = \sum_{i=0}^{m} w_{ji}(n) y_i(n) \tag{3.3}$$

Finalmente, um conjunto de saída é produzido como a resposta atual da rede – y – que é subtraída da resposta desejada – d – produzindo o sinal do erro (ε), definido na Equação 3.4:

$$\varepsilon(n) = \frac{1}{2} \sum_{j} \left(d_j(n) - y_j(n)\right)^2 \tag{3.4}$$

Na etapa de retropropagação (*backward*), iniciando na camada de saída, o gradiente local do neurônio é calculado, os pesos são ajustados de acordo com a regra delta generalizada (HAYKIN, 2001). O ajuste dos pesos – w – procura minimizar a diferença entre a saída calculada pela rede e a saída desejada, ou seja, a minimização do erro da

resposta atual da rede. A forma genérica, utilizando a regra delta, para atualização dos pesos é apresentada na Equação 3.5:

$$w_j(n) = w_j(n) + \eta\varepsilon(n)x_j(n) \tag{3.5}$$

em que $x_j(n)$ é a entrada para o neurônio j e o η é a taxa de aprendizagem do algoritmo de retropropagação. Essa taxa é responsável pelo grau de transformação que o vetor de pesos será submetido, sendo que uma taxa com valor muito baixo torna o aprendizado lento, enquanto que um valor muito alto provoca oscilações no treinamento podendo impedir a convergência do processo. A inclusão do termo *momentum* na regra delta é uma forma de aumentar a influência da taxa de aprendizagem evitando o problema de instabilidade (BRAGA et al., 2000):

$$\Delta w_{ji}(n) = \alpha\Delta w_{ji}(n-1) + \eta\sigma_j(n)y_i(n) \tag{3.6}$$

sendo que α corresponde ao termo *momentum*. Para que o ajuste $\Delta w_{ji}(n)$ convirja, o termo *momentum* deve ficar no intervalo entre 0 e 1.

A complexidade computacional de um algoritmo é normalmente medida em termos de número de multiplicações, adições e armazenamentos envolvidos na sua implementação. Diz-se que um algoritmo de aprendizagem é computacionalmente eficiente quando a sua complexidade computacional é polinomial em relação ao número de parâmetros ajustáveis que devem ser atualizados de uma iteração para a seguinte. Neste sentido, pode-se dizer que o algoritmo de retropropagação é computacionalmente eficiente. Especificamente, quando o usamos para treinar um perceptron de múltiplas camadas contendo um total de W pesos sinápticos, incluindo os níveis de viés, a sua complexidade computacional é linear em W. Portanto, a conclusão é que a complexidade computacional do algoritmo de retropropagação é linear em relação a W, ou seja, $O(W)$ (HAYKIN, 2001).

A arquitetura de RNA é definida a partir de alguns parâmetros: número de camadas da rede, número de neurônios em cada camada, tipo de função de ativação. No que diz respeito ao número de camadas, a única certeza é que deve haver uma entrada e uma camada de saída de modo a ser capaz de apresentar e obter dados de e para a RNA e o número de neurônios em cada uma dessas duas camadas é especificado pelo número de parâmetros de entrada e saída que são usados para modelar cada problema. Portanto, para um bom desempenho da rede, definir o número de camadas ocultas e o número de neurônios em cada camada oculta é de extrema importância, principalmente no que se refere a capacidade de generalização. É sabido que o nú-

mero de neurônios está diretamente ligado a habilidade que a rede tem de resolver problemas de certa complexidade. Consequentemente, a determinação do número de neurônios é, na verdade, o problema mais fundamental no desempenho de redes neurais e motiva boa parte das pesquisas na área (BRAGA et al., 2000).

3.2 Configuração automática de Rede Neural Artificial

Rede neural é um conceito de extrema importância na computação, uma vez que é responsável pela solução de muitos problemas complexos. Devido ao forte vínculo existente entre a definição da arquitetura de uma RNA e a qualidade da solução obtida, a busca pela arquitetura ótima torna-se um dos aspectos mais importantes nessa área, isso se justifica não apenas pelo fato de estar diretamente associado ao desempenho do modelo, mas também porque não há nenhum conhecimento teórico sobre a forma como essa arquitetura será encontrada ou como essa arquitetura deveria ser (BENARDOS; VOSNIAKOS, 2007). O método mais usado é o empírico, baseado na experiência e na observação, porém, é um processo demorado e necessita da experiência de um especialista em RNA e no problema a ser tratado, envolvendo assim um alto grau de incerteza.

É dito que o projeto de uma rede neural utilizando o algoritmo de retropropagação é mais uma arte do que uma ciência, visto que a definição dos numerosos fatores envolvidos no projeto são o resultado de experiência particular de cada um (HAYKIN, 2001). Mas pesquisas recentes de RNA estão justamente tratando desse assunto. Apesar do aumento da atividade de pesquisa, o problema de encontrar uma arquitetura ótima ainda não foi respondido de forma definitiva. No entanto, as diferentes abordagens existentes podem ser categorizadas do seguinte modo (BENARDOS; VOSNIAKOS, 2007):

(i) métodos empíricos ou estatísticos que são usados para estudar o efeito dos parâmetros internos de uma RNA e escolher valores apropriados para eles com base no desempenho do modelo;

(ii) métodos híbridos tais como inferência neuro difusa, no qual a RNA pode ser interpretada como um sistema *fuzzy* adaptativo ou pode operar em *fuzzy* em vez de números reais;

(iii) algoritmos construtivos e/ou de poda – *pruning*– que, respectivamente, adicionam e/ou removem neurônios de uma arquitetura inicial usando um critério previamente especificado para indicar como o desempenho de RNA

é afetado pelas mudanças;

(iv) estratégias evolucionárias que pesquisam sobre o espaço de arquiteturas, no qual, cada ponto corresponde uma possível solução, variando o número de camadas ocultas e número de neurônios ocultos através da aplicação de operadores genéticos e avaliação das diferentes arquiteturas de acordo com uma função objetivo.

Portanto, o problema correspondente a encontrar uma arquitetura de rede neural ótima pode ser formulado como um problema de otimização, isto é, um processo de busca de argumentos que resultam num valor extremo de uma ou mais funções que representem o problema em questão. A otimização dispõe-se a determinar uma configuração ótima de um dado sistema, não necessitando percorrer todas as possíveis soluções. A definição de uma solução "ótima" está intrinsecamente ligada ao problema considerado, ao método adotado e as tolerâncias utilizadas.

A palavra "ótimo" é apontada como "máximo" ou "mínimo" dependendo das circunstâncias, "ótimo" é um termo técnico que implica medição quantitativa, da mesma forma que a palavra "otimizar"significa atingir um ótimo. A teoria da otimização é o ramo da matemática que engloba o estudo quantitativo do ótimo e dos métodos para encontrá-lo (ANTONIOU; LU, 2007).

Aplicando o conceito de otimização no problema de definição de uma arquitetura ótima em RNA, por meio da associação do valor de desempenho da RNA a cada ponto ou solução de tal modo que este valor está apoiado em algum critério de otimalidade – complexidade, menor erro quadrático ou velocidade de treinamento – é possível construir uma superfície na qual os pontos mais altos (ou mais baixos) equivalem as melhores arquiteturas, determinando assim um espaço de busca (CARVALHO, 2011b).

Espaço de busca é um conjunto que contém todas as soluções do problema. Esse espaço pode ser finito ou infinito, enumerável ou não enumerável e deve conter apenas soluções viáveis, isto é, aquelas que obedecem às restrições do problema tratado (BECCENERI, 2008). No contexto de RNA, o espaço de busca consiste em todas as diferentes combinações de camadas ocultas e neurônios ocultos, isto é, de todas as arquiteturas.

Em problemas de otimização, utiliza-se uma função objetivo – f_{obj} – e um conjunto de restrições que devem ser satisfeitas. Na configuração da arquitetura de uma RNA,

a função objetivo empregada é uma combinação de dois fatores: diferença quadrática entre o valor desejado (alvo) e a saída de rede neural, e um fator de penalidade, em decorrência da complexidade da arquitetura da rede. Esta função objetivo que foi definida por Carvalho (2011b) e é descrita na Equação 3.7

$$f_{obj} = penalidade \times \left(\frac{\rho_1 \times E_{trein} + \rho_2 \times E_{gen}}{\rho_1 + \rho_2} \right) \tag{3.7}$$

em que ρ_1 e ρ_2 são termos que modificam a relevância atribuída ao erro de generalização e de treinamento, tornando a avaliação da função objetivo flexível, devido ao fato de que o erro de treinamento está diretamente ligado a capacidade de memória da rede e o erro de generalização refere-se a habilidade da RNA em identificar padrões que são similares aos usados na fase de treinamento (CARVALHO, 2011b). Foi então adotado os seguintes valores para $\rho_1 = 1$ e $\rho_2 = 0,1$, os mesmos utilizador por Carvalho (2011a). Quando $\rho_1 > \rho_2$ dá-se maior relevância na capacidade de memorização da rede com relação aos padrões apresentados em detrimento da generalização. Os erros de treinamento E_{trein} e generalização E_{gen} são obtidos utilizando a Equação 3.4.

O fator penalidade, expresso por Equação 3.8, auxilia na busca por arquiteturas de RNA leves, ou seja, arquiteturas com menor número de camadas ocultas e de neurônios em suas camadas, evitando assim o sobre-ajuste – *overfitting* – dos dados de treinamento da rede (HAYKIN, 2001).

$$penalidade = c_1 e^{(n_{neurônios})^2} + c_2(n_{épocas}) + 1 \tag{3.8}$$

em que $c_1 = 5 \times 10^8$ e $c_2 = 5 \times 10^5$, são parâmetros de ajuste para encontrar um equilíbrio da complexidade baseados na número de neurônios ($n_neurônios$) e número de épocas (n_epocas).

O valor mínimo de f_{obj} corresponde a uma arquitetura simples, em que número total de pesos e viés na etapa de aprendizagem sejam reduzidos, exibindo um comportamento consistente no espaço de soluções aliado a baixos erros de treinamento e generalização (CARVALHO, 2011b).

Um otimizador é empregado para variar cinco parâmetros:

(i) número de camadas intermediárias;

(ii) número de neurônios em cada camada intermediária (oculta);

(iii) função de ativação;

(iv) taxa de aprendizagem η;

(v) constante de *momentum* α

sendo que os três primeiros parâmetros são considerados discretos e os dois últimos são considerados contínuos. O intervalo de valores permitidos para estes parâmetros é mostrado na Tabela 3.1.

Tabela 3.1 - Parâmetros utilizados no MPCA para definir arquiteturas de RNAs.

Tipo	Parâmetros	Valores
Discreto	$n_{\text{camadas\_ocultas}}$	$[1 \dots 3]$
	$n_{\text{neurônios}}$	$[1 \dots 32]$
	$\{f_{\text{ativação}}\}$	{tanh (1)}
		{sigmoide (2)}
		{gaussiana (3)}
Contínuo	η	$[0,0 \dots 1,0]$
	α	$[0,1 \dots 0,9]$

Um conjunto de soluções candidatas que correspondam a uma arquitetura de RNA é gerada pelo otimizador. Para cada solução, a RNA é ativada e o processo de treinamento é iniciado, até que um critério de parada seja satisfeito (mínimo erro ou número total de épocas). Com os valores obtidos pela RNA, a função objetivo é calculada e os parâmetros da RNA são atualizados. Este processo se repete até que um valor ótimo ou próximo do ótimo para a função objetivo seja encontrado.

3.3 Algoritmo de Colisão de Múltiplas Partículas

Problemas de otimização são encontrados em diversos campos: ciência, engenharia, economia, etc. Um problema de otimização fundamenta-se em obter uma boa combinação dentre um conjunto de variáveis para maximizar ou minimizar uma função, geralmente chamada de função objetivo. Esses problemas podem ser classificados (BECCENERI, 2008) em:

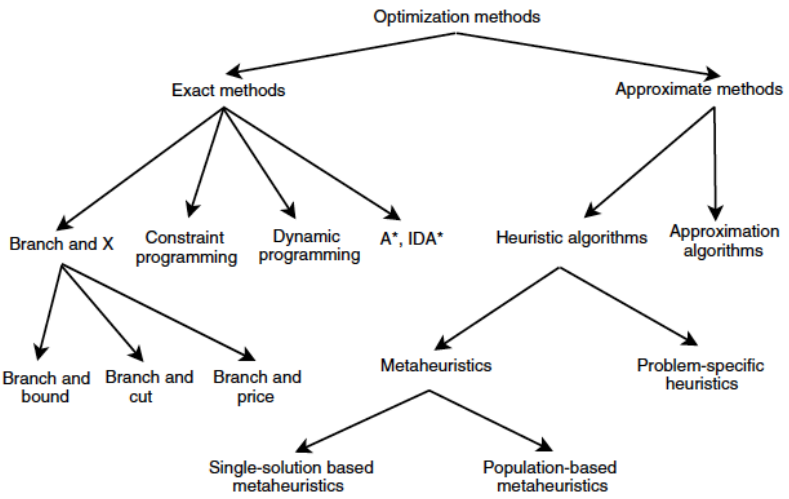
- otimização contínua: variáveis assumem valores reais (ou contínuos);
- otimização combinatória ou discreta: variáveis assumem valores discretos (ou inteiros);

- otimização mista: variáveis inteiras e contínuas.

O principal objetivo de um problema de otimização é encontrar uma solução ótima global, ou todas as soluções ótimas globais. Existem diferentes famílias de modelos de otimização que são utilizadas para formular e resolver problemas de tomada de decisão (TALBI, 2009).

Os métodos de otimização podem ser classificados em: métodos exatos ou métodos aproximados, determinísticos ou não-determinísticos. Os métodos exatos produzem o mesmo resultado sempre que o valor de entrada é repetido, soluções ótimas são obtidas, porém o custo computacional pode ser demasiadamente alto para determinados problemas, tornando-se inviável. Os métodos aproximados – estocásticos – podem, diante de uma mesma entrada do problema, produzir resultados diferentes, pois consideram no seu processamento algum evento pseudo-aleatório, geram soluções de alta qualidade em um tempo razoável para uso prático, porém não há garantia de encontrar uma solução ótima global. A Figura 3.7 é um esquema

Figura 3.7 - Métodos clássicos de otimização



Fonte: (TALBI, 2009)

Dos métodos de otimização aproximada, a família das metaheurísticas ganharam muito popularidade nas últimas décadas, estão entre as técnicas mais promissoras e bem sucedidas, pois fornecem soluções "aceitáveis" em um tempo razoável para resolver problemas difíceis e complexos. A palavra heurística tem sua origem na

antiga palavra grega *heuriskein*, que significa a arte de descobrir novas estratégias para solucionar problemas. O sufixo meta, também uma palavra grega, significa "metodologia de nível superior" (TALBI, 2009). De grande importância na utilidade de uma metaheurística é o balanço dinâmico entre diversificação – *exploration* – e intensificação – *exploitation*, este balanço é essencial para prevenir que a busca fique estacionada em um ótimo local, habilitando a busca pelo ótimo global (LUZ, 2012)

O avanço significativo na computação de alto desempenho, alguns desses sistemas são capazes de fornecer *petaflops*, permite a solução de grandes problemas através de novos algoritmos que podem tirar proveito destes ambientes. Com isso, as metaheuristicas também ganharam espaço, sendo desenvolvidas e adaptadas para esses ambientes.

Um exemplo desses novos algoritmos, desenvolvidos para ambientes computacionais de alto desempenho, é o Algoritmo de Colisão de Múltiplas Partículas (MPCA), um método de otimização estocástico desenvolvido por (LUZ, 2012), que utiliza as vantagens do balanço dinâmico. Sua eficiência foi demonstrada nos seguintes trabalhos: *MPCA Metaheuristics for automatic architecture optimization of a supervised artificial neural network* (SAMBATTI et al., 2012b), uma aplicação na previsão do clima sazonal de mesoescala, *Automatic configuration for neural network applied to atmospheric temperature profile identification* (SAMBATTI et al., 2012a), aplicado na identificação dos perfis de temperatura verticais obtidos a partir de dados de satélites, *New learning strategy for supervised neural network: MPCA meta-heuristic approach* (ANOCHI et al., 2003), prever o clima mensal de mesoescala para o campo de precipitação.

O MPCA é uma extensão do *Particle Collision Algorithm* (PCA) que foi proposto por Sacco e Oliveira (2005), um método de otimização estocástico baseado no paradigma de recozimento simulado, onde uma solução pode ser aceita com uma certa probabilidade, o que garante a convergência para o ótimo global. A semelhança do PCA com o recozimento simulado está na estrutura: uma configuração inicial é escolhida, é feita a avaliação. As qualidades das duas configurações são comparadas: nova e antiga, e é tomada uma decisão sobre a configuração nova – aceitável ou não – se for, ela tomará o lugar da configuração antiga para a próxima etapa, esta aceitação pode evitar a convergência para um ótimo local (SACCO; OLIVEIRA, 2005).

O PCA foi inspirado no espalhamento de uma partícula em um reator nuclear, enfatizando, além do comportamento de espalhamento, o comportamento de absorção, que ocorre quando uma partícula atinge um núcleo com baixo valor da função obje-

tivo e é absorvida. Caso contrário, uma partícula com alto valor da função objetivo é espalhada para outra região ao se chocar com um núcleo . Isso permite que o espaço de busca do problema seja amplamente percorrido e que as regiões mais promissoras sejam exploradas através de eventos sucessivos de espalhamento e absorção.

O algoritmo PCA tem sido utilizado com sucesso em diversos casos de otimização, de problemas de testes a problemas de aplicações reais (SACCO; OLIVEIRA, 2005) (SACCO et al., 2009). Sua estrutura de funcionamento baseia-se na escolha de uma solução inicial (*Old-Config*), que é modificada por uma perturbação estocástica(*Perturbation*{.}), levando à construção de uma nova solução (*New-Config*). A nova solução é comparada (função *Fitness*{.}), e pode ou não ser aceita. Se a nova solução não é aceita, o processo de espalhamento (*Scaterring*{.}) é usado. A exploração em torno de posições mais próximas é garantida usando as funções *Perturbation*{.} e *Small-Perturbation*{.}. Se a nova solução é melhor que a anterior, esta nova solução é absorvida. Se uma solução pior é achada, a partícula pode ser enviada para uma posição diferente no espaço de busca, permitindo que o algoritmo escape do mínimo local (LUZ, 2012). A Figura 3.8 mostra o principal trecho do algoritmo PCA:

Figura 3.8 - Algoritmo PCA

```
Gera uma solução inicial: Old-Config
Best-Fitness = Fitness(Old-Config)
Para n = 0 até # de iterações
    Perturbation(.)
        Se Fitness(New-Config) > Fitness(Old-Config)
            Se Fitness(New-Config) > Best-Fitness
                Best-Fitness = Fitness(New-Config)
            Fim-se
            Old-Config = New-Config
            Exploration(.)
        Senão
            Scattering(.)
        Fim-se
Fim-Para
```

Fonte: Luz (2012)

Para garantir que as pertubações impostas ao algoritmo não levem à construção de uma solução, na qual um de seus pontos esteja fora do espaço válido de buscas, uma

verificação destes limites é realizada constantemente para evitar esse contratempo, tal como descrito na Figura 3.9:

Figura 3.9 - Função para pertubação no PCA.

```
Pertubation(.)
    Para i = 0 até (Dimensões−1)
        Superior = Limite-superior[i]
        Inferior = Limite-inferior[i]
        Aleatório = Random(0, 1)
        New-Config[i] =Old-Config[i] + ((Superior − Old-Config[i] ∗ Aleatório) -
       ((Old-Config[i] − Inferior)∗(1 − Aleatório))
            Se New-Config[i] > Superior
                New-Config[i] = Limite-superior[i]
             Senão
               Se New-Config[i] > Inferior
                   New-Config[i] = Limite-inferior[i]
               Fim-se
            Fim-se
    Fim-Para
Retorna
```

Na função exploração (*Exploration*), pequenas pertubações estocásticas são aplicadas à solução através da função (*Small-Pertubation*) com objetivo de executar uma pequena exploração local, a fim de verificar a existência de uma possível solução ainda melhor de uma determinada vizinhança, como apresentada na Figura 3.10.

Figura 3.10 - Função para exploração no PCA.

```
Exploration(.)
    Para n = 0 até # de iterações
        Small-Perturbation(.)
       Se Fitness(New-Config) > Fitness(Old-Config)
           Old-Config = New-Config
        Fim-se
    Fim-Para
Retorna
```

A Figura 3.11 demonstra o procedimento de espalhamento (*Scattering*), um esquema

estocástico que tenta evitar a prisão do algoritmo em uma região de ótimo local no espaço de buscas.

Figura 3.11 - Função para espalhamento no PCA.

```
Scattering(.)
    Pscattering = 1 - Fitness(New-Config) / Best-Fitness
    Se Pscattering > Random(0, 1)
        Old-Config = Solução aleatória
    Senão
        Exploration(.)
    Fim-Se
Retorna
```

Os parâmetros que regulam o funcionamento do algoritmo PCA são: número de iterações do algoritmo, número de iterações da busca local, tamanho do raio de pertubação, tamanho do raio de pertubação da busca local e a função de probabilidade usada no cálculo do espalhamento.

O algoritmo MPCA é similar ao PCA, porém foi introduzido o conceito e o uso de múltiplas partículas, equivalente a uma população de soluções candidatas. Portanto, com um conjunto de n partículas a exploração é feita de forma independente e colaborativa no mesmo espaço de soluções, com isso torna-se necessário a implementação de um mecanismo de comunicação entre as partículas. Uma técnica, denominada *blackboard* foi adotada, a informação *Best-Fitness*, que corresponde ao melhor resultado obtido pelas partículas, é compartilhada entre todas envolvidas no processo de busca. Este processo foi implementado usando o *Message Passing Interface* (MPI) para a aplicação em máquinas com memória distribuída (LUZ, 2012).

O pseudo-código para o MPCA é apresentado na Figura 3.12.

Figura 3.12 - MPCA: pseudo-codigo.

```
Solução inicial gerada: Old-Config
Best-Fitness = Fitness(Old-Config)
Atualiza Blackboard
Para n = 0 até # de partículas
    Para n = 0 até # iterações
    Atualiza Blackboard
    Perturbation(.)
        Se Fitness(New-Config) > Fitness(Old-Config)
            Se Fitness(New-Config) > Best-Fitness
                Best-Fitness = Fitness(New-Config)
            Fim-se
            Old-Config = New-Config
            Exploration(.)
        Senão
            Scattering(.)
        Fim-se
    Fim-para
Fim-para
```

Os parâmetros que regulam o funcionamento do MPCA são (LUZ, 2012).:

- número de iterações globais do algoritmo;
- número de iterações para as perturbações locais, correspondendo ao laço e à quantidade de chamadas a função (Small-Perturbation);
- função de probabilidade (*Pscattering*) usada na ativação do espalhamento;
- limite superior (LS) e limite inferior (LI) da intensidade de perturbação na busca local, usada pela função (Small-Perturbation);
- número de processadores (*Nprocessadores*) usados;
- número de partículas (*Nparticulas*) para explorar o espaço de busca;
- número de iterações (*Nblackboard*) executadas até que a sincronização (*blackboard*) seja ativada.

Outra característica relevante do MPCA é a capacidade de escape de ótimos locais. A busca cooperativa, ou seja, a troca de informação entre as partículas, auxilia o processo de convergência, uma vez que o processo de espalhamento leva em consi-

deração a informação que está sendo compartilhada entre as partículas. A principal vantagem do MPCA está no aproveitamento do poder computacional provido por múltiplos processadores na tarefa de explorar o espaço de soluções através da troca de informação, que é peça central deste mecanismo melhorado de escape de ótimos locais. Luz (2012) apresentou melhoras significativas em precisão e tempo computacional na resolução de problemas de otimização continua com uso do algoritmo MPCA, garantindo bons resultados.

3.3.1 MPCA na otimização da RNA

No MPCA, cada partícula está estruturada por um vetor de soluções que correspondem aos seguintes parâmetros: *atualP* é a partícula atual; *novaP* é uma nova partícula; $NAFO$ é o número de avaliações da função objetivo; a melhor partícula em cada iteração fica armazenada em *melhorP* e $N_{experimentos}$ armazena o número total de experimentos. LI e LS correspondem oas limites inferior e superior, respectivamente, estão diretamente ligados a intensidade de busca da perturbação local, e os parâmetros LB e UB são vetores que contém os valores mínimo e máximo para cada variável.

O algoritmo MPCA para RNAs é apresentado na Figura 3.13. Cada partícula está estruturada em um vetor de soluções de tamanho D, sendo este a dimensão do problema (número de parâmetros otimizados) e Fitness é um número real com o valor da função objetivo.

Figura 3.13 - MPCA: pseudo-codigo.

```
Divide os dados em subconjuntos de treinamento e validação
Define os parâmetros de controle do MPCA:
nProcessadores; nPartículas; nBlackboard; lI; lS; lB; uB
Para i ← 1, nProcessadores faça
    Para j ← 1, nPartículas faça
        atual p_{i,j}.Solução ← solução Aleatória
        Treinar a RNA para atualP.Solução
    Fim Para
Fim Para
Para i ← 1, nProcessadores faça
    melhorP_i ← atualizarBlackBoard(atualP_{i,j})
Fim Para
NAFO ← 0
ultimaAtualização ← 0
Enquanto NAFO < NAFO_max faça
   Para i ← 1, nProcessadores faça
       Para j ← 1, nPartículas faça
          novaP_{i,j} ← Perturbação(AtualP_{i,j}.Solução)
          Se novaP_{i,j}.Fitness < atualP_{i,j}.Fitness então
             atualP_{i,j} ← novaP_{i,j}
             atualP_{i,j} ← Exploração(atualP_{i,j}, novaP_{i,j}, melhorP_i)
          Senão
             atualP_{i,j} ← Espalhamento(atualP_{i,j}, novaP_{i,j}, melhorP_i)
          Fim Se
          Se atualP_{i,j}.Fitness < melhorP_{i,j}.Fitness então
             melhorP_{i,j} ← atualP_{i,j}
          Fim Se
       Fim Para
       Se (NAFO - últimaAtualização) > nBlackboard então
          melhorP_i ← AtualizarBlackBoard(atualP_{i,j})
          últimaAtualização ← NAFO
       Fim Se
   Fim Para
Fim Enquanto Para i ← 1, nProcessadores faça
    melhorP_i ← AtualizarBlackBoard(atualP_{i,j})
Fim Para
Devolver melhorP_i
```

4 COMPUTAÇÃO EM FPGA

Existem duas maneiras distintas de executar a computação: *hardware* e *software*. O computação em *hardware*, como os circuitos integrados de aplicação especifica – *Application Specific Integrated Circuits* (ASICs), são projetados exclusivamente para executar uma determinada computação e, portanto, são muito rápidos e eficientes ao efetuar o cálculo para o qual foram concebidos. No entanto, o circuito não pode ser alterado após a fabricação e, caso seja realmente necessário realizar qualquer modificação de parte de seu circuito, um novo projeto e uma nova fabricação do *chip* será inevitável, um processo com custo altamente elevado.

Na computação em *software*, uma solução muito mais flexível, os processadores realizam um conjunto de instruções para executar uma computação. Alterando as instruções do *software*. A funcionalidade do sistema é alterada sem alterar o *hardware*. No entanto, a desvantagem desta flexibilidade é que o desempenho é muito inferior ao de um ASIC. O processador deve ler cada instrução da memória, decodificar o seu significado e somente então executá-lo. O conjunto de instruções que pode ser usado por um programa é determinado no tempo de fabricação do processador. Quaisquer outras operações a serem implementadas devem ser construídas a partir de instruções existentes (HAUCK; DEHON, 2008).

A demanda por desempenho computacional continua no mesmo ritmo nos últimos anos, porém manter a lei de *Moore*, que diz que a capacidade de processamento dos *chips* irá dobrar a cada dezoito meses, tornou-se um grande desafio. A limitação física dos *chips* atingiu a capacidade máxima (curva de desempenho versus a dissipação de calor gerada). Tornou-se praticamente impossível continuar entregando mais capacidade sem uma mudança de conceito e de arquitetura. Algumas soluções foram testadas, como os *chips multicore*. Porém, a necessidade de computadores mais potentes para executar aplicações mais complexas continuou inabalável (RAISCH, 2010).

Varias alternativas para atender a necessidade por mais desempenho computacional foram testadas, como por exemplo o aumento do nível de paralelismo entre os diversos núcleos de processamento de um mesmo *chip*, necessitando de novos conceitos de programação para explorar este tipo de arquitetura. Surgiu então uma nova abordagem conhecida por sistemas híbridos.

Os sistemas híbridos não dependem exclusivamente de um maior número de processadores de propósito geral, este termo é aplicado quando se usa um conjunto

heterogêneo de mecanismos de computação – componentes – podendo ser construídos usando qualquer número de componentes de computação, incluindo os tradicionais processadores de propósito geral (GPP), multiprocessadores de *chips* homogêneos ou heterogêneos (CMPs), *Field Programmable Gate Array* (FPGAs), *field programmable object arrays* (FPOAs), unidades de processamento gráfico (GPUs), processadores de sinais digitais (DSPs), processadores de instruções específicas de aplicações (ASIPs), processadores *many-cores* de uso mais restrito (MIC: Many Integrated Cores). Sendo que cada um dos componentes tem seus atributos e suas próprias características únicas, tanto em termos de arquitetura quanto no processo de desenvolvimento de aplicativos (CHAMBERLAIN et al., 2007).

Estes sistemas são constantemente utilizados em aplicações de computação embarcada de alto desempenho (HPEC) onde há frequentemente restrições de tamanho, peso e energia no sistema. Além disso, como é comum com os processadores de propósito geral, grandes coleções de "nós" híbridos podem ser conectadas em rede para formar um *cluster*, obtendo vantagens significativas na área de computação de alto desempenho (HPC). Para muitas aplicações de computação intensiva, um sistema híbrido pode ter um desempenho significativamente maior do que um *cluster* de tamanho semelhante construído exclusivamente com processadores de propósito geral (CHAMBERLAIN et al., 2007).

O uso adequado de cada componente está estreitamente ligado com a aplicação tratada. As GPUs – processadores gráficos – dependem da utilização de um grande número de núcleos de processamento para lidar com uma alta carga de computação com alto grau de paralelismo de dados. Por sua vez, dispositivos lógicos programáveis, fornecem melhores formas para explorar padrões específicos de paralelismo, além de ser possível configura-los dinamicamente. Sistemas híbridos correspondem a estratégia que utiliza vários dispositivos arquiteturalmente diferentes em um único fluxo de trabalho, permitindo que cada dispositivo execute as tarefas às quais está mais adaptado. Uma aplicação desta técnica é a utilização de GPU para o processamento gráfico deixando a CPU (unidade de processamento central) livre para o restante da aplicação. A melhoria no desempenho surge quando parte da aplicação é executada em uma unidade especializada (GOMES, 2012).

Uma subárea da computação híbrida é a computação híbrida reconfigurável que utiliza dispositivos lógicos programáveis combinados com processadores de propósito geral para a computação de aplicações intensivas. Os dispositivos lógicos programáveis fornecem formas para explorar padrões específicos de paralelismo de dados e

de controle. Além de possibilitarem a configuração dinâmica do seu funcionamento aplicada a diferentes estruturas de *hardware*, permitindo o desenvolvimento mais flexível de processadores específicos para cada tipo de aplicação. Por estas características, dispositivos lógicos programáveis têm ganhado popularidade na comunidade de alto desempenho nos últimos anos (GOMES, 2012).

A fim de obter esses benefícios de desempenho e ainda suportar uma ampla gama de aplicações, os sistemas reconfiguráveis são geralmente formados por uma combinação de dispositivo reconfigurável e microprocessador de propósito geral. O processador executa as operações que não podem ser feitas de forma eficiente pelo computação reconfigurável, como o controle dependente de dados e possivelmente os acessos de memória, enquanto os núcleos computacionais são mapeados para o *hardware* reconfigurável, que podem ser compostos por FPGAs ou *hardware* configurável personalizado (COMPTON; HAUCK, 2002).

A computação reconfigurável (RC) está se estabelecendo rapidamente como uma grande disciplina que abrange vários assuntos de aprendizagem, incluindo ciência da computação e engenharia eletrônica. Foi apresentada como acelerador em uma variedade de aplicações. Como por exemplo, a criptografia de dados. Sendo capaz de alavancar tanto o paralelismo como a manipulação de dados. Outras aplicações recentes que foram usando *hardware* reconfigurável incluem: reconhecimento de padrões, algoritmo genético para o problema do caixeiro viajante, etc. (TODMAN et al., 2005; COMPTON; HAUCK, 2002).

Computação Reconfigurável refere-se à prática de utilizar dispositivos de *hardware* reconfiguráveis para acelerar o desempenho computacional de um sistema para uma aplicação particular. Tem sido impulsionada em grande parte pelo desenvolvimento de FPGAs, que são dispositivos que combinam os benefícios do *hardware* e do *software*. Esse tipo de computação implementa circuitos como *hardware*, fornecendo enorme poder, área e benefícios de desempenho em relação ao *software*, mas com uma peculiaridade, eles podem ser reprogramados de forma barata e fácil para implementar uma ampla gama de tarefas. Esses sistemas podem ser centenas de vezes mais rápidos do que os projetos baseados em microprocessadores. No entanto, ao contrário de ASICs, esses cálculos são programados no *chip*, não permanecem congelados pelo processo de fabricação. Isso significa que um sistema baseado em FPGA pode ser programado e reprogramado muitas vezes (HAUCK; DEHON, 2008).

No entanto, a fusão dos benefícios de *hardware* e *software* tem um custo. FPGAs fornecem quase todos os benefícios da flexibilidade de *software* e modelos de desen-

volvimento, e quase todos os benefícios da eficiência de *hardware* - mas não completamente. Comparado a um microprocessador, esses dispositivos são tipicamente várias ordens de magnitude mais rápidas e mais eficientes, mas criar programas eficientes para eles é mais complexo. Os FPGAs são úteis somente para operações que processam grandes fluxos de dados, tais como processamento de sinal, rede e similares. Um projeto em ASIC pode levar de meses a anos e com um custo multimilionário, enquanto um projeto de FPGA pode levar apenas dias para criar com um custo bem menor. A simplicidade de desenvolvimento de um FPGA e a capacidade de corrigir facilmente problemas e atualizar a funcionalidade tornam-os uma alternativa de projeto atraente (HAUCK; DEHON, 2008).

4.1 FPGA - *Field Programmable Gate Array Architecture*

Dispositivos lógicos programáveis surgem como uma solução intermediária entre as duas formas de computar: *software* e *hardware*. Fazem parte dessa categoria os dispositivos como *Programmable Array Logic* (PAL), *Generic Array Logic* (GAL), *Complex Programmable Logic Device* (CPLD) e *Field Programmable Gate Array* (FPGA). Destes dispositivos, o FPGA é o dispositivo que possui maior flexibilidade devido à abundância e ao tamanho reduzido de suas unidades básicas, o que permite a configuração de sistemas complexos com um custo muito menor quando comparado a um ASIC (*Application-Specific Integrated Circuit* (HAUCK; DEHON, 2008)

Um FPGA é um *chip* de silício contendo uma matriz de blocos lógicos configuráveis (CLBs). Ao contrário de um Circuito Integrado Específico de Aplicação (ASIC) que pode executar uma única função específica durante o tempo de vida do *chip*, um FPGA pode ser reprogramado para executar uma função diferente em questão de microssegundos. Antes de ser programado, um FPGA não sabe nada sobre como se comunicar com os dispositivos que o rodeiam, permitindo uma grande flexibilidade em sua utilização, mas a complexidade da programação aumenta absurdamente. A capacidade de reprogramar FPGAs os levou a serem amplamente utilizados por projetistas de *hardware* para protótipos de circuitos. Nos últimos anos, FPGAs começaram a conter recursos suficientes para torná-los de interesse para a comunidade HPC. Recentemente, os fornecedores de hardware HPC começaram a oferecer soluções que incorporam FPGAs em sistemas HPC, onde eles podem atuar como co-processadores, acelerando assim os *kernels* chaves dentro de um aplicativo. A Cray Inc. foi uma das primeiras empresas a produzir tal sistema com o XD1. SGI emergiu rapidamente como o principal concorrente da Cray (WAIN et al., 2006).

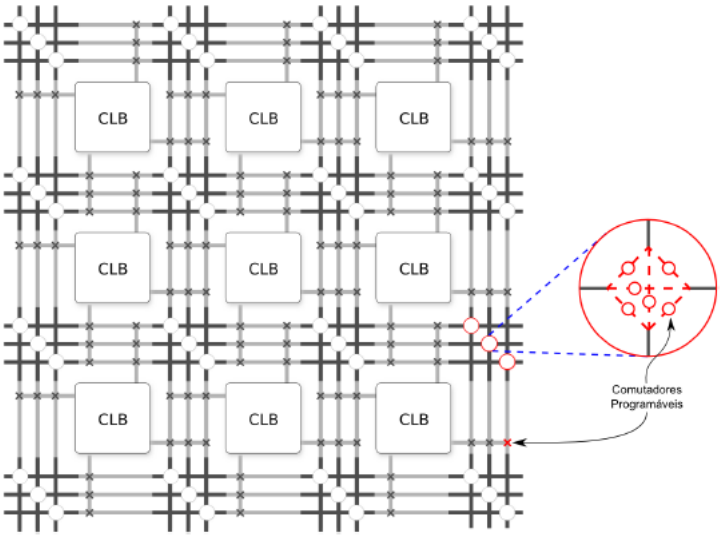
O recente aumento no interesse FPGA da comunidade HPC chegou num momento

em que microprocessadores convencionais estão lutando para manter-se com a lei de Moore (MOORE, 1965). Este abrandamento dos ganhos de desempenho nos microprocessadores, juntamente com o aumento do custo de manutenção das suas exigências de energia consideráveis levou a um maior interesse em qualquer tecnologia que pudesse oferecer uma alternativa rentável. O uso de FPGAs em sistemas HPC pode fornecer três vantagens distintas em relação aos *clusters* de computação convencionais (WAIN et al., 2006).:

- os FPGAs consomem menos energia do que os microprocessadores convencionais;
- usando FPGAs como aceleradores podem aumentar significativamente a densidade de computação;
- FPGAs podem fornecer um aumento significativo no desempenho de um determinado conjunto de aplicações.

Sua utilização pretende alcançar a flexibilidade de soluções baseadas em *software* atrelado ao desempenho de soluções implementadas em *hardware* (CHAMBERLAIN et al., 2007; SKLIAROVA; FERRARI, 2003).

Figura 4.1 - Arquitetura padrão de FPGA



Fonte: (SKLIAROVA; FERRARI, 2003)

Conforme pode ser visto na Figura 4.1, o FPGA consiste em um grande arranjo de

células lógicas ou blocos lógicos configuráveis (CLBs) contidos em um único circuito integrado. Cada célula contém capacidade computacional para implementar funções lógicas e *flips-flops* para a realização de lógica sequencial, e realizar roteamento para comunicação entre elas, permitindo a configuração de ligações arbitrárias, de tal forma que os elementos lógicos possam ser conectados de maneira apropriada (GOMES, 2012).

Uma das maiores vantagens das FPGAs é a sua arquitetura flexível para uma ampla gama de aplicações. Estas aplicações incluem implementação de controladores de dispositivos, circuitos de codificação, lógica arbitrária, prototipagem e emulação de sistemas, entre outras. As FPGAs recentes passaram a incorporar várias estruturas heterogêneas tais como blocos de memória, o que possibilita a implementação de sistemas completos num único encapsulamento.

O primeiro FPGA comercialmente viável foi desenvolvido pela Xilinx em 1984 e entrou para *National Inventors Hall of Fama* por essa realização em 2006. FPGAs foram inicialmente utilizados para a lógica discreta, mas expandiu sua área de aplicação ao processamento de sinais, a computação de alto desempenho embarcada , e recentemente como aceleradores para computação de alto desempenho.(BRODTKORB et al., 2010).

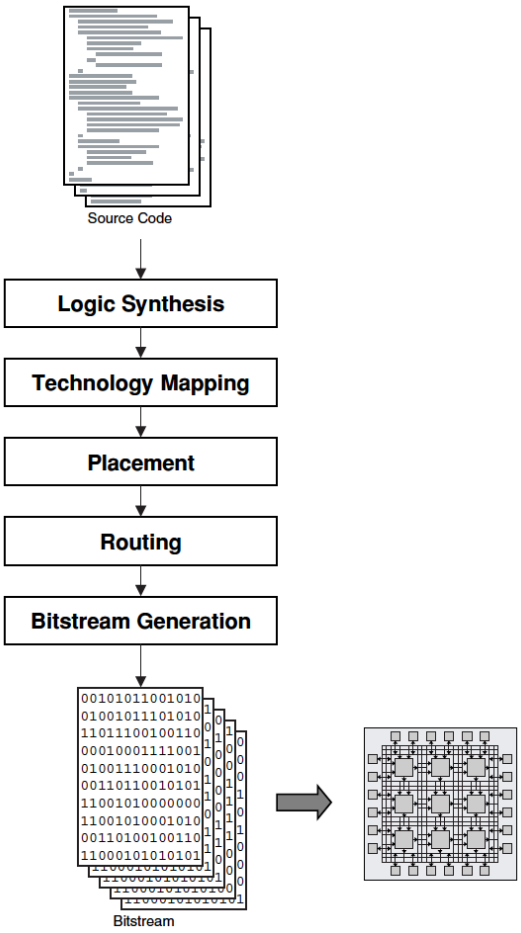
Na área de alto desempenho, os FPGAs vêm propiciando um aumento na capacidade computacional superior ao obtido com processadores de uso geral, por permitirem a criação de arquiteturas massivamente paralelas e especializadas. Entre as aplicações que fazem uso desta tecnologia estão: criptografia de dados, aplicações financeiras e computação científica (MENOTTI, 2010).

O desenvolvimento de aplicações para FPGAs é tradicionalmente feito através de linguagens de descrição de *hardware* (HDL – *Hardware Description Languade*). Verilog e VHDL (*VHSIC – Very High Speed Integrated Circuit*) são linguagens que permitem a descrição em alto nível do comportamento lógico da aplicação e do fluxo interno de dados (GOMES, 2012). O uso de VHDL é vantajoso pela flexibilidade de implementação, por ser uma linguagem padrão (IEEE – *Institute of Eletrical and Eletronic Engineer*) e independente da ferramenta de desenvolvimento. Deste modo, uma mesma descrição de projeto pode ser utilizada em diversas arquiteturas de dispositivos (SHIGUEMORI, 2007).

A conversão da descrição de *hardware* até a configuração do FPGA segue as etapas ilustradas na Figura 4.2. O primeiro passo é a descrição do *hardware* em VHDL

ou Verilog. Este código fonte é sintetizado, transformando a lógica em alto nível em portas lógicas interconectadas. Na sequência, o processo de mapeamento separa as portas lógicas em grupos para serem melhor adaptadas aos recursos lógicos do FPGA. O quarto passo é o posicionamento, que avalia em qual bloco lógico cada grupo de portas lógicas deve ser configurado. Por fim, no roteamento, são determinadas as interconexões que irão transportar os sinais entre os blocos. Ao final destas etapas é gerado um *bitstream* que contém as informações necessárias para configurar o FPGA com a aplicação projetada (HAUCK; DEHON, 2008).

Figura 4.2 - Fluxo de desenvolvimento para FPGAs.



Fonte: (HAUCK; DEHON, 2008)

A programação para sistemas híbridos reconfiguráveis é uma tarefa desafiadora, por causa do duplo paradigma enfrentado para a implementação de aplicações para este tipo de arquitetura. O programador precisa assumir o papel de projetista de

hardware, sendo necessário conhecer os detalhes técnicos do sistema híbrido alvo para aproveitar todos os recursos disponíveis (CARDOSO et al., 2010b; GOMES, 2012).

Acredita-se amplamente que a principal barreira à adoção de tecnologia reconfigurável promissora é a falta de fluxos de programação adequados que ofereçam um nível de abstração mais elevado do que o atualmente oferecido pelos HDL disponíveis. Ferramentas que suportam especificações de programação de alto nível acelerariam enormemente o ciclo de desenvolvimento de sistemas reconfiguráveis e facilitariam a migração de algoritmos já desenvolvidos para esses sistemas - um aspecto-chave para sua ampla aceitação (CARDOSO et al., 2010a).

Durante a última década um grande número de sistemas de computação reconfigurável foi desenvolvido pela comunidade de pesquisa, que demonstrou a capacidade de atingir alto desempenho para um conjunto selecionado de aplicações.(CARDOSO et al., 2010b).

4.2 Cray XD1

Neste trabalho, utilizou-se o sistema híbrido reconfigurável de alto desempenho Cray XD1, disponível no Laboratório Associado de Computação e Matemática Aplicada do Instituto Nacional de Pesquisas Espaciais (LAC-INPE). O sistema Cray XD1 é um dos primeiros sistemas comerciais de computação de alto desempenho (HPC) a incluir FPGAs como aceleradores programáveis pelo usuário que estão localizados próximos da memória principal da CPU (unidade central de processamento) (ULMER et al., 2005).

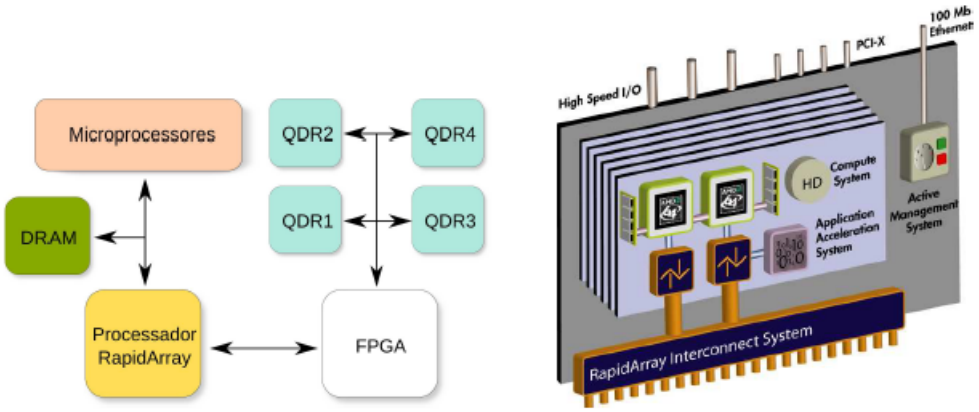
O Cray XD1 é um sistema híbrido reconfigurável de alto desempenho, sua arquitetura abriga seis nós de computação (*blades*) interconectados por uma rede de alta velocidade chamada *Rapid Array*, cada *blade* fornece dois processadores AMD Opteron 64 bits e um FPGA Xilinx Virtex II Pro.

Para permitir a comunicação entre o FPGA e os processadores de um *blade*, a Cray disponibiliza a API *RapidArray Transport Core*. Essa API permite que a CPU realize envio e recebimento de dados com FPGA e que o FPGA leia e escreva em regiões de memória compartilhadas com a CPU. No Cray XD1, a FPGA tem acesso a diferentes níveis de memória. A DRAM (*Dynamic random access memory*), possui até 8 *gigabytes* (GB) e latência variável de leitura. O QDR II SRAM, com 4 blocos independentes de 4 Mb, pode ser acessada diretamente pelo FPGA com latência de 8 ciclos. Por fim, em menor quantidade, está disponível a memória interna ao FPGA

que pode ser acessada em 1 ciclo.

A arquitetura de uma *blade* do Cray XD1 pode ser vista na Figura 4.3(a), enquanto que a Figura 4.3(b) apresenta a visão lógica de um nó desse sistema. É possível observar que o dispositivo reconfigurável tem acesso direto a quatro bancos de memória QDR II SRAM, os quais possuem 4 Megabits (MB) cada. O processador *RapidArray* é o elemento responsável por gerenciar a comunicação entre FPGA e CPU. Esse recurso permite que os processadores enviem dados para o FPGA e que o FPGA leia dados da DRAM. No desenvolvimento de aplicações híbridas, existem duas questões chave que devem ser observadas: o uso eficiente dos diferentes níveis de memória disponíveis no sistema (CARDOSO et al., 2010a) e a transferência de dados entre os dispositivos (GOMES et al., 2011).

Figura 4.3 - Visão lógica e arquitetura de um *blade* do *Cray* XD1



(a) (b)

Fonte: (CRAY..., 2005)

A plataforma Cray XD1 conecta FPGAs diretamente a CPUs usando seu mecanismo de interconexão proprietário RapidArray. O FPGA pode ser configurado para mapear componentes SRAM externos para o espaço de memória do usuário do processador, permitindo que um aplicativo de *software* transfira dados diretamente para o FPGA. Sob condições operacionais ideais, o FPGA pode processar dados em uma região da SRAM, enquanto a CPU simultaneamente preenche ou drena *buffers* em outra região. Uma biblioteca de rotinas de software com suporte para hardware FPGA pré-projetado é fornecida aos usuários finais do sistema para aceleração de aplicativos.

4.2.1 Hierarquia da Memória

O FPGA, em um *blade*, acessa dois tipos diferentes de memória, como pode ser visto na Figura 4.3(a): blocos de QDR e memória DRAM. Porém ele também tem disponível blocos de memória conhecidos como *Block RAMS*. Possuem formas de acesso e quantidades específicas para cada recurso.

Sendo que a DRAM é o mais alto nível, com a maior quantidade de memória disponível e com latência de leitura não constante, e pode ser acessada pelo *RapidArray Transport Core*, interface de comunicação disponibilizada pela Cray.

Os bancos de memória QDR II SRAM com 4 MB cada estão em um nível mais baixo, a latência de acesso a esses bancos é de 8 ciclos para a leitura, e o acesso é feito utilizando a interface QDR II SRAM *Core*, disponibilizada pelo fabricante. Essa interface permite gravações e leituras completamente independentes nesses bancos de memória. Outra característica dos FPGAs Virtex II Pro é que possuem bancos de memória internos que podem ser utilizados diretamente pelas aplicações com gravações e leitura em um único ciclo de relógio. Usualmente, a DRAM é usada para compartilhar dados da aplicação em *software* com o FPGA. Esses dados são carregados para a QDR II SRAM para serem acessados durante a execução da aplicação em *hardware*. A memória interna do FPGA, disponível em menor quantidade, é utilizada para registradores e cache de dados. Cada aplicação necessita de recursos e formas de acesso diferentes, sendo um importante fator para diminuir tempos de execução (GOMES et al., 2011).

4.2.2 Comunicação: CPU–FPGA

Cray fornece um núcleo de comunicação para o FPGA que permite que os dados sejam transferidos entre a memória principal e o FPGA. Do ponto de vista do usuário do FPGA, esse núcleo é composto por duas interfaces separadas: uma para transferências iniciadas pelo *host* e outra para transferências iniciadas pelo FPGA. Cada interface possui portas para transações de leitura e gravação. A interface de transferência iniciada pelo *host* é relativamente simples de usar porque os circuitos do usuário de FPGA simplesmente precisam aceitar dados de gravação de entrada e gerar respostas para as solicitações de leitura de entrada. Enquanto as leituras envolvem o uso de identificadores de *tags* para correlacionar respostas de leitura a solicitações de leitura, essas *tags* podem ser gerenciadas com registros de atraso simples (ULMER et al., 2005).

A integração entre CPU e FPGA é um importante desafio que deve ser levado em consideração durante o desenvolvimento de aplicações nesta tecnologia, porque a maneira pela qual os FPGAs são inseridos na arquitetura de um sistema dita a taxa na qual os FPGAs podem trocar dados com outros recursos do sistema. Historicamente, o caminho mais comum para integrar recursos FPGA em uma estação de trabalho tem sido através do uso de dispositivos periféricos *add-on cards*. Esses cartões facilitam a comunicação entre o FPGA e o processador por meio de interfaces de *E/S* padrão, porém seu baixo desempenho de comunicação torna desafiador implementar um sistema onde há um acoplamento apertado entre aceleradores FPGA e processadores *host* (HAUCK; DEHON, 2008).

As transferências iniciadas por FPGA são ligeiramente mais complexas porque os circuitos FPGA do usuário são responsáveis pela orquestração das transferências. Além de fornecer o endereço físico da memória do *host* utilizado na transferência (ULMER et al., 2005). Para a comunicação entre dispositivos em um blade, a Cray disponibiliza a *API RapidArray Transport Core* (RTCore), que é um componente que deve ser usado com a descrição do algoritmo em VHDL. Esta entidade é composta por duas interfaces denominadas *Fabric Request* e *User Request* (CRAY..., 2005). As Figuras 4.4 e 4.5 apresentam, respectivamente, os sinais dessas interfaces. Esses dois conjuntos de sinais permitem duas formas de comunicação, onde ou a CPU ou o FPGA é o responsável pela iniciativa de começar a transferência de dados (GOMES et al., 2013).

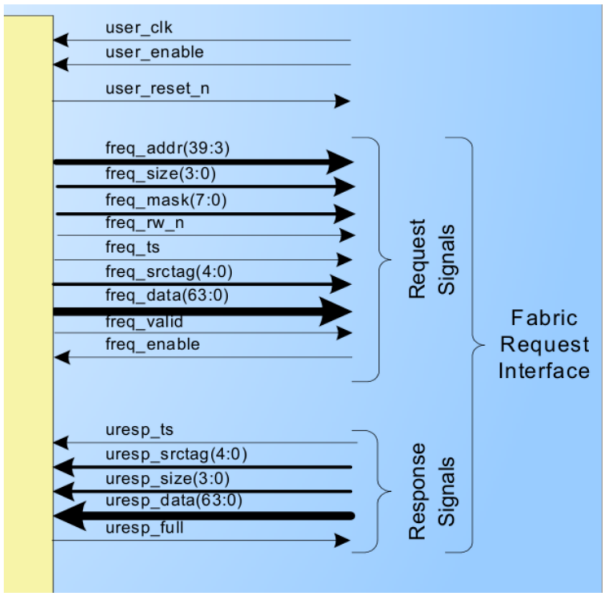
A interface *Fabric Request*, que realiza a comunicação usando uma abordagem *push*, permite que o programa executado nos processadores envie e requisite dados para o FPGA. Esta abordagem mantém a CPU ocupada durante a transferência dos dados. Para permitir esse tipo de comunicação, a Cray disponibiliza a biblioteca em linguagem C *enlib*, a qual abstrai ao programa o FPGA como um arquivo. Deste modo, a transferência de dados entre os processadores e o FPGA é realizada através de leituras e gravações pelo programa neste "arquivo". Com a realização de uma leitura ou escrita, o FPGA recebe, através do *RTCore*, uma requisição que deve ser tratada pela aplicação do FPGA. Em caso de leitura, deverá ser retornado um valor ao *RapidArray Transport Core* para que ocorra o retorno da função chamada pelo programa em linguagem de alto nível. Somente é permitida a manipulação de um *quadword* (64 bits) por requisição utilizando a interface *Fabric Request* (CRAY..., 2005). Na Figura 4.4 é possível ver os sinais que são ativados quando uma requisição é feita pelo programa *Request Signals* e os sinais que são usados pela aplicação em VHDL para responder essas requisições.

A abordagem que utiliza a interface *User Request* é conhecida como *pull* e, diferentemente da *Fabric Request*, mantém os processadores livres durante a transferência de dados entre o programa em C e o FPGA. Para isso, esta interface permite que o FPGA realize leituras e escritas em um espaço de memória DRAM compartilhado pelo programa. O endereço para a região de memória, que é compartilhada utilizando a *einlib*, precisa ser enviado ao FPGA através do bloco *Fabric Request*.

Com o endereço disponível, a aplicação descrita para o FPGA é capaz de fazer até 32 requisições sequenciais ao *RTCore*. Cada requisição pode solicitar até 8 posições contíguas da memória do programa.Os retornos das solicitações ao *RTCore* não são necessariamente na ordem em que foram realizadas. O *RTCore* garante somente a ordem das 8 posições contíguas de cada requisição (CRAY..., 2005). O sistema descrito para o FPGA deve ordenar os dados através do auxílio de *tags* disponibilizadas durante a requisição e o retorno.

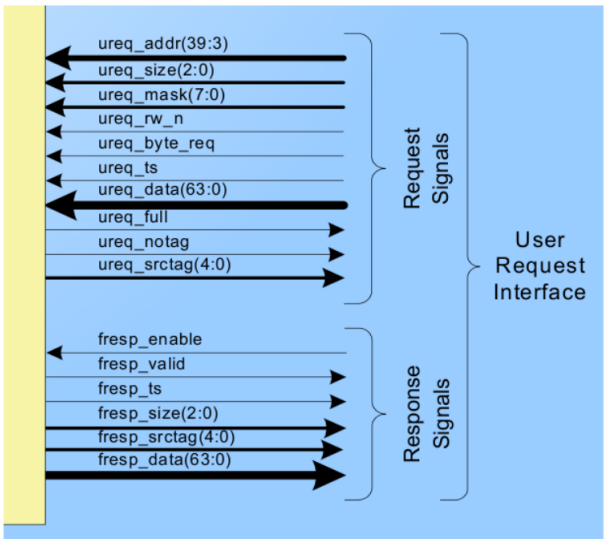
Outra solução é aguardar o retorno de uma solicitação antes de realizar outra. Os sinais da interface *User Request* podem ser vistos na Figura 4.5. Através dos sinais *Request Signals* o FPGA é capaz de requisitar posições de memória, sendo atendidas através dos *Response Signals*.

Figura 4.4 - Sinais do bloco Fabric Request



Fonte: (CRAY..., 2005)

Figura 4.5 - Sinais do bloco User Request



Fonte: (CRAY..., 2005)

4.3 Implementação de Rede Neural em FPGA

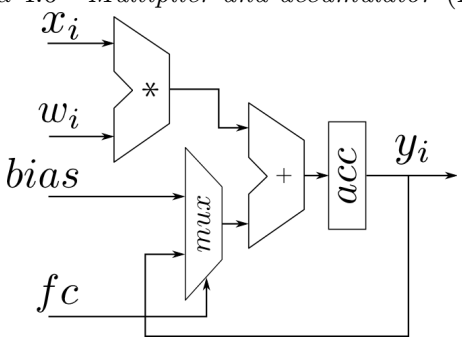
O sucesso alcançado em aplicações pelas redes neurais motivou e motiva a implementação deste tipo de máquina de computação em *software*, em *chips* e em *hardware* reconfigurável.

Durante os anos 80 houve um esforço para projeto e implementação de neurocomputação em *hardware*. Inicialmente os projetos eram focados em desenvolvimento de *chips* ASIC (Application Specific Integrated Circuits). Mas, não havia uma demanda de mercado significativa para atrair interesse da indústria de forma estável e os projetos foram abandonados. Nesta época, a tecnologia de FPGAs não estava muito desenvolvida. Porém, com a evolução tecnológica, a capacidade de processamento, memórica e redução de custo, permite retomar a idéia de implementar redes neurais artificiais em dispositivos de *hardware* (OMONDI; RAJAPAKSE, 2006)

A implementação da rede neural perceptron de múltiplas em FPGA pode explorar ao máximo o paralelismo inerente desse dispositivo. O projeto é estruturado em componentes de computação, permitindo uma melhor organização da arquitetura. A unidade MAC (*Multiplier and Accumulator*), núcleo do processamento digital em FPGA, é uma extensão o multiplicador básico presente em grande parte dos microprocessadores, eles são os blocos principais do processador e tem um grande

impacto na velocidade do processador, ver Figura 4.6 – no qual é armazenado o resultado do produto das entradas pelos pesos sinápticos, somando o viés.

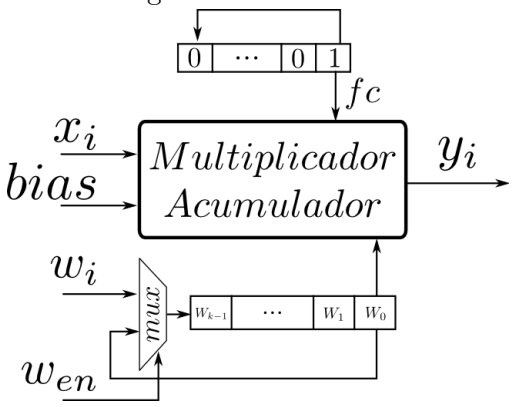
Figura 4.6 - *Multiplier and accumulator* (MAC)



Cada entrada x_i é multiplicada por um peso w_i e somado ao valor do registrador *acc*, mas dependendo do valor do sinal *fc*, que é sinalizado a cada primeiro ciclo, o viés também será somado ao resultado da multiplicação antes de ser armazenado, esta ação permite que o primeiro produto da multiplicação seja adicionado ao viés, e os demais ao valor já acumulado, não necessitando zerar o acumulador no início de cada cálculo de atividade interna, evitando ciclos extras de computação (GOMES et al., 2013). O código VHDL pode ser visto no Anexo A.

O próximo módulo é o neurônio artificial, unidade básica de uma rede neural, ele faz uso de um MAC e uma estrutura de controle, conforme pode ser visto no diagrama da Figura 4.7, associado a elementos lógicos que fazem o controle dessa estrutura. Neste módulo, a entrada x_i e o viés são conectados diretamente ao componente MAC e o sinal (fc) que indica o primeiro ciclo de operação é gerado por meio de registrador de deslocamento que possui um único bit posicionado inicialmente a direita. Esse registrador tem o mesmo número de bits que o total de dados de entrada que são computados pelo neurônio. Tal abordagem permite que o neurônio seja reiniciado automaticamente sem a inclusão de estruturas lógicas complexas (GOMES et al., 2013).

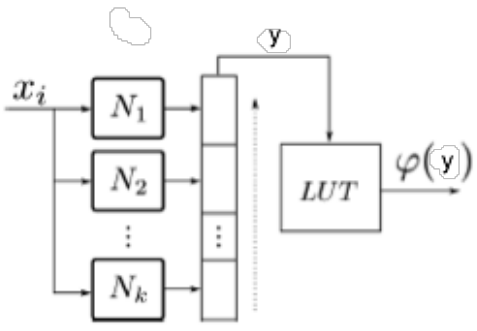
Figura 4.7 - Neurônio



Para gerenciar os pesos, registradores interconectados em um fila circular são usados. Os pesos são deslocados a direta a cada nova entrada x_i. A configuração dessa estrutura é realizada através de w_i e da sinalização de w_en, fazendo com que os pesos possam ser alterados conforme necessário, aumentando a flexibilidade dessa solução. A organização dos pesos sinápticos nessa estrutura permite que o neurônio retorne a sua configuração inicial depois que todas as entradas forem calculadas, permitindo a computação de fluxos de dados contínuos sem a necessidade de reinicialização e de estruturas de controle complexas (GOMES et al., 2013) (VHDL no Anexo A).

O último módulo computacional – camada da rede neural – é um combinação de vários módulos de neurônios com as entradas conectadas por um barramento único. A saída de cada neurônio está conectado a uma posição de um registrador de deslocamento, ver Figura 4.8. Os neurônios podem receber dados, e os resultados (saídas) são encaminhados individualmente para uma *lookup table* (LUT): esta operação simula a função de ativação (GOMES et al., 2013).

Figura 4.8 - Implementação de uma RNA: *pipeline*



O projeto de MLP em FPGA é formado por meio da concatenação serial de camadas formando uma rede neural artificial. A entrada de cada camada está diretamente conectada com a saída da camada anterior. Considerando que uma camada é um módulo de computação, um pipeline de operações sequencias é implementado. A computação para cada camada pode ser executado de forma independente, permitindo que múltiplos conjuntos de dados possam ser calculados com um pequeno atraso sequencial para cada camada executada.

O FPGA tem comportamento determinístico. Com base nessa afirmação, o total de ciclos necessários para computar um conjunto de dados de uma camada da rede neural é dado por (GOMES et al., 2011):

$$\text{ciclos} = (n \times \text{Entrada} + 5) + (n \times \text{Neurônio} + 8) \tag{4.1}$$

no qual $\{n\text{Entrada}\}$ corresponde ao número total de entradas usadas na rede neural, no caso de uma aplicação em assimilação de dados esse número será constante e igual a 2, e $\{n\text{Neurônio}\}$ representa o número total de neurônios utilizados em cada camada. O total de ciclos necessários para executar uma rede neural completa é a soma dos ciclos de cada uma de suas camadas.

5 MODELOS DE PREVISÃO

Modelos matemáticos descritos por um conjunto de equações diferenciais parciais e condições inicial e de contorno em um domínio específico são utilizados para descrever muitos fenômenos geodinâmicos. Esses modelos são capazes de fazer a ligação entre as características causais de um processo geodinâmico com os seus efeitos (ISMAIL-ZADEH; TACKLEY, 2010).

Modelos oceânicos são descritos por equações de movimento com forças distribuídas, condições iniciais e de contorno, levando também em conta a distribuição de probabilidade dos erros nessas duas últimas quantidades. No caso mais simples, as médias e as covariâncias serão prescritas com a hipótese Gaussiana. Essas equações que descrevem o modelo são resolvidas por aproximação numérica, produzindo campos de circulação oceânica através do domínio do modelo em um dado intervalo de tempo. Sendo que comparações entre os resultado do modelo e de dados observacionais podem ser feitos, quando este último está disponível. É possível que os dados de entrada do modelo sejam ajustados para satisfazer os dados experimentais. Em resumo, o modelo oceânico de Chua e Bennett (2001) é uma hipótese nula para a distribuição de probabilidade dos erros nas componentes do modelo oceânico *forward* (CHUA; BENNETT, 2001).

5.1 Equação da Onda 1D

Esta equação diferencial da onda de primeira ordem foi uma das equações diferenciais descritas e analisadas no artigo de Courant et al. (1928). O artigo de Courant–Friedrichs–Lewy é considerado por muitos como o primeiro a estruturar a disciplina de análise numérica e estabelecer critérios para convergência de soluções numéricas — o critério é hoje conhecido como CFL (Courant–Friedrichs–Lewy), um acrônimo em homenagem aos três matemáticos alemães.

Seguindo (BENNETT, 2002), a metodologia de assimilação de dados será aplicada a equação da onda de primeira ordem (HOFFMAN; FRANKEL, 2001)

$$\frac{\partial \eta_F}{\partial t} + c\frac{\partial \eta_F}{\partial x} = F(x,t) \tag{5.1}$$

nesta equação, η é o deslocamento – valor a ser estimado pela assimilação de dados, c é a constante da velocidade de fase (ou relação de dispersão), $F(x,t)$ é a forçante externa, t é o tempo e x é a coordenada espacial. O subíndice F em η_F indica a solução avançada, *a priori*. Graficamente, à medida que t aumenta, a função inicial

$\eta(x,0)$ move-se para a direita com velocidade c. Tendo como condição inicial:

$$\eta(x,0) = I(x) \tag{5.2}$$

para $0 < x < L$, em que I é especificado. E a condição de contorno:

$$\eta(0,t) = B(t) \tag{5.3}$$

em que $t > 0$ e $B(t)$ é especificado.

A condição inicial utilizada na integração da equação da onda (Equação 5.1) é a solução analítica da equação KdV (Korteweg-deVries) avaliada em t, descrita como:

$$\eta(x,t) = \eta_0 \frac{1}{\cosh^2[(x - vt)/\Delta]} \tag{5.4}$$

no qual η_0 é a amplitude do *sóliton* (onda solitária que mantém sua forma e se propaga em velocidade constante), v é a velocidade de fase ($v = c + \alpha\eta_0/3$) e Δ é o tamanho da escala do *sóliton* – $\Delta = \sqrt{(12\beta)/(\alpha\eta_0)}$. Os parâmetros utilizados na integração do modelo é mostrado na Tabela 5.1. A evolução temporal ocorre em 2000 passos de tempo com 128 pontos na coordenada x para a Equação 5.1.

Tabela 5.1 - Parâmetros utilizados na integração do modelo (Fonte: Furtado (2012))

Parâmetro	Valor
η_0	-60 m
c	2,42 ms^{-1}
α	$-1,62 \times 10^{-2}$ s^{-1}
β	1,46$\times 10^5$ m^{3}s-1
Δ	1340 m
v	2,75 ms^{-1}

O modelo dado pela Equação 5.1 foi integrado como método de Crank Nicholson (LYNCH, 2004; ISMAIL-ZADEH; TACKLEY, 2010) com derivada espacial de quarta ordem:

$$\begin{aligned} \eta_p^{k+1} = \eta_p^k \quad &- \quad c_1(\eta_{p-2}^{k+1} + 8\eta_{p-1}^{k+1} - 8\eta_{p+1}^{k+1} + \eta_{p+2}^{k+1}) - c_1(\eta_{p-2}^{k} + 8\eta_{p-1}^{k} - 8\eta_{p+1}^{k}) \\ &+ \quad \frac{1}{2}(F_p^{k+1} + F_p^k) \end{aligned} \tag{5.5}$$

na qual $c_1 = c\Delta t/(24\Delta x)$, F é o termo forçante do modelo, $p = 1, ..., n_x$ e $k =$

$1, ..., n_t$. Rearranjando a Equação 5.5, obtém-se:

$$\begin{aligned} c_1\eta_{p-2}^{k+1} - c_1 8\eta_{p-1}^{k+1} + \eta_p^{k+1} + c_1 8\eta_{p+1}^{k+1} - c_1\eta_{p+2}^{k+1} &= -c_1\eta_{p-2}^{k} + c_1 8\eta_{p-1}^{k} + \eta_p^{k} \\ -c_1 8\eta_{p+1}^{k} + c_1\eta_{p+2}^{k} &+ \frac{1}{2}(F_p^{k+1} + F_p^{k}) \ . \end{aligned} \tag{5.6}$$

Para solucionar a Equação 5.6, é necessário resolver o sistema linear da seguinte forma:

$$A\eta^{k+1} = B\eta^{k} \ . \tag{5.7}$$

em que A e B são matrizes associadas a versão discreta do modelo.

No processo de assimilação de dados, um número finito de observações coletadas dentro de um domínio espacial ($0 \leq x \leq L$) e temporal ($0 \leq t \leq T$) é assumido. As observações, designadas por d_m, são medidas pontuais e imperfeitas das variáveis independentes $\eta(x,t)$ coletadas em M pontos no espaço e no tempo (x_m, t_m) e são dadas por:

$$d_m = \eta(x_m, t_m) + \epsilon_m \tag{5.8}$$

em que $1 \leq m \leq M$, $\eta(x,t)$ é o campo de deslocamento real e desconhecido e ϵ é o erro na medida das observações. Não se pode esperar que o modelo seja perfeitamente consistente com os dados pois a forçante, a condição inicial e os dados contêm erros, portanto

$$\eta_F(x,t) \neq d_m, \quad 1 \leq m \leq M \ . \tag{5.9}$$

Analogamente, tem-se no caso discreto

$$d_m = \eta_{p_m}^{k_m} + \epsilon_m, \quad 1 \leq m \leq M \ . \tag{5.10}$$

Devido aos erros no cálculo *a priori* para a forçante (F), a condição inicial (I) e a condição de contorno (C), a circulação real deve satisfazer:

$$\frac{\partial \eta_F}{\partial t} + c\frac{\partial \eta_F}{\partial x} = F(x,t) + f(x,t) \tag{5.11}$$

$$\eta(x,0) = I(x) + i(x) \tag{5.12}$$

$$\eta(0,t) = C(t) + c(t) \tag{5.13}$$

em que $0 \leq x \leq L$, $0 \leq t \leq$, $f(x,t)$ é o erro na forçante F, $i(x)$ é o erro em I e $c(t)$ é o erro em C. A condição de contorno é periódica no tempo com $\eta(0,t) = \eta(L,t)$ com $0 \leq t \leq T$. Consequentemente, as Equações 5.8, 5.11, 5.12 e 5.13 correspondem

aos *residuais* de observação, modelagem, condição inicial e condição de contorno, respectivamente.

5.2 Sistema de água rasa 2D

O sistema de águas rasas é usado quando a profundidade é muito menor do que o domínio horizontal do movimento. Modelos oceânicos representados por sistema de águas rasas são convencionalmente formulados como equações de movimento com forçante distribuída, com condições iniciais e de contorno apropriadas, podendo ser utilizados para prever a velocidade da água e seu nível em vários pontos em uma região do fluido em diferentes instantes de tempo (SAMPSON, 2008; RANDALL, 2006).

A propagação de perturbações na água e a evolução de outros fluidos incompressíveis em resposta à aceleração gravitacional e rotacional é descrita pelas equações de água rasa. O período de tempo de interesse para um modelo que descreve marés é de 12 a 24 horas enquanto que para um modelo de tsunamis é de 15 a 30 minutos (SAMPSON, 2008).

A conservação de massa e momento resultam no sistema de água rasa – conjunto de equações diferenciais parcial hiperbólicas derivadas das equações de Navier-Stokes de dinâmica dos fluidos. As equações são resolvidas por uma aproximação numérica, produzindo campos de circulação oceânica em todo o domínio do modelo durante um intervalo de tempo. A equação de onda linearizada bidimensional é descrita na Equação 5.16, tendo como variáveis independentes a profundidade do fluido (H) e o campo de velocidade do fluido bidimensional (u e v). A força gravitacional (g) é a única força que agirá sobre o fluido.

$$\frac{\partial u}{\partial t} - fv + g\frac{\partial q}{\partial x} + r_u u = F_u \tag{5.14}$$

$$\frac{\partial v}{\partial t} + fu + g\frac{\partial q}{\partial y} + r_v v = F_v \tag{5.15}$$

$$\frac{\partial q}{\partial t} + H\left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y}\right) + r_q q = 0 \tag{5.16}$$

em que $0 < x < X$, $0 < y < Y$, f representa o parâmetro de Coriolis, (r_u, r_v, r_q) são os coeficientes de amortecimento, u e v são componentes da velocidade, F_u e F_v são forçantes externas, H é a profundidade média do oceano e q é a perturbação de superfície livre. Se $q \equiv q'$, o oceano está em equilíbrio hidrostático ou em estado de equilíbrio.

As condições iniciais consideradas são dadas por:

$$u(x, y, 0) = I^u(x, y) = 0 \tag{5.17}$$

$$v(x, y, 0) = I^v(x, y) = 0 \tag{5.18}$$

$$q(x, y, 0) = I^q(x, y) = 0 \tag{5.19}$$

e as condições de contorno são

$$u(x + X, y, t) = u(x, y, t) \tag{5.20}$$

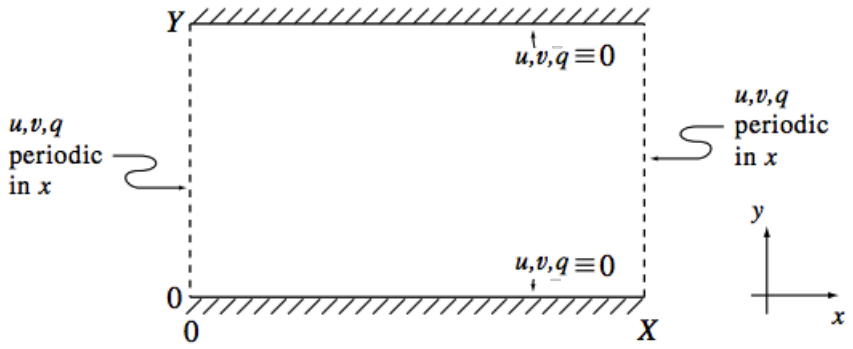
$$v(x + X, y, t) = v(x, y, t) \tag{5.21}$$

$$q(x + X, y, t) = q(x, y, t) \tag{5.22}$$

$$q(x, y + Y, t) = q(x, y, t) \tag{5.23}$$

Para a discretização do modelo, foi utilizado a grade-C de Arakawa para a discretização espacial, utilizando o método *forward-backward* para discretização temporal (MESINGER; ARAKAWA, 1976). Tem-se paredes norte e sul rígidas na condição de contorno, ou seja, $u(x, 0, t) = u(x, Y, t) = v(x, 0, t) = v(x, Y, t) = 0$ e todos os campos são periódicos na direção x (ver Figura 5.1).

Figura 5.1 - Canal periódico com paredes rígidas.



(BENNETT, 2002)

As forçantes consideradas no modelo são $F_u = -(C_d \rho_a u_a^2)/(H\rho_w)$ e $F_v = 0$, sendo C_d é coeficiente de arrasto, ρ_a é a densidade do ar, ρ_w é a densidade da água e u_a é o vento zonal. As condições de contorno rígidas são dadas por

$$v_{i,1}^k = 0 \;\; e \;\; v_{i,NJ}^k = 0 \; . \tag{5.24}$$

As condições de contorno periódicas são assumidas para as demais variáveis:

$$\begin{aligned}\Phi(0,j) &= \Phi(NI,j)\\ \Phi(1,j) &= \Phi(NI+1,j)\\ \Phi(i,0) &= \Phi(i,NJ)\\ \Phi(i,1) &= \Phi(i,NJ+1)\end{aligned} \tag{5.25}$$

onde $\Phi = u, v, q$.

O modelo dado pelas Equações 5.14, 5.15 e 5.16 é discretizado em diferenças finitas:

$$\frac{u_{i,j}^{k+1}-u_{i,j}^{k}}{\Delta t} - f\left(\frac{v_{i,j+1}^{k}+v_{i,j}^{k}+v_{i-1,j+1}^{k}+v_{i-1,j}^{k}}{4}\right) + g\left(\frac{q_{i,j}^{k+1}-q_{i-1,j}^{k+1}}{\Delta y}\right) + r_u u_{i,j}^{k} = F_{u_{i,j}^{k}} \tag{5.26a}$$

$$\frac{v_{i,j}^{k+1}-v_{i,j}^{k}}{\Delta t} - f\left(\frac{u_{i+1,j}^{k}+u_{i,j}^{k}+u_{i+1,j-1}^{k}+u_{i,j-1}^{k}}{4}\right) + g\left(\frac{q_{i,j}^{k+1}-q_{i,j-1}^{k+1}}{\Delta y}\right) + r_v v_{i,j}^{k} = F_{v_{i,j}^{k}} \tag{5.26b}$$

$$\frac{q_{i,j}^{k+1}-q_{i,j}^{k}}{\Delta t} + H\left(\frac{u_{i+1,j}^{k}-u_{i,j}^{k}}{\Delta x} + \frac{v_{i,j+i}^{k}-v_{i,j}^{k}}{\Delta y}\right) + r_q q_{i,j}^{k} = 0 \tag{5.26c}$$

$$\tag{5.26d}$$

Os parâmetros utilizados na integração do modelo de água rasa 2D se encontram na Tabela 5.2. Com esses parâmetros, foi possível reproduzir o experimento de (BENNETT, 2002).

A Figura 5.2 ilustra a discretização espacial do método de grade-C de Arakawa. E a integração no tempo é realizada pelo esquema *forward-backward* (MESINGER; ARAKAWA, 1976):

$$u_{ij}^{n+1} = u_{ij}^{n} + \Delta t\, p_u\{u_{ij}^{n}, v_{ij}^{n}, q_{ij}^{n}, (F_u)_{ij}^{n}\} \tag{5.27}$$

$$v_{ij}^{n+1} = v_{ij}^{n} + \Delta t\, p_v\{u_{ij}^{n}, v_{ij}^{n}, q_{ij}^{n}, (F_u)_{ij}^{n}\} \tag{5.28}$$

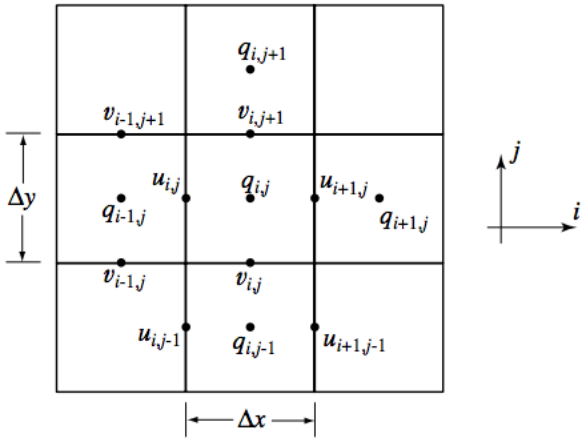
$$q_{ij}^{n+1} = q_{ij}^{n} + \Delta t\, p_u\{u_{ij}^{n+1}, v_{ij}^{n+1}, q_{ij}^{n}, (F_q)_{ij}^{n}\} \tag{5.29}$$

Tabela 5.2 - Parâmetros utilizados na integração do modelo (Fonte: Furtado (2012))

Parâmetro	Valor
H	5000 m
T	$1,8 \times 10^4$ s
g	$9,806$ m s^{-2}
f	$1,0 \times 10^{-4}$ s^{-1}
r_u	$1,8 \times 10^4$ s^{-1}
r_v	$1,8 \times 10^4$ s^{-1}
r_q	$1,8 \times 10^4$ s^{-1}
C_d	$1,6 \times 10-3$
ρ_a	1,275 kg m^{-3}
ρ_w	$1,0 \times 10^3$ kg m^{-3}

em que $\partial\alpha/\partial t \approx (\alpha^{n+1} - \alpha^n)/\Delta t$ com $(\alpha = u, v, q)$ e $p_\alpha\{.\}$ está agrupando todos os termos restantes nas Equações (5.27), (5.28), (5.29) – aqui: $F_q = 0$.

Figura 5.2 - Discretização com grade-C de Arakawa para diferenças finitas espaciais.



(BENNETT, 2002)

6 RESULTADOS E DISCUSSÕES

O propósito desta tese de doutorado é aplicar a metodologia de assimilação de dados utilizando rede neural implementada em FPGA em modelos oceanográficos simplificados: equação da Onda 1D e sistema de água rasa 2D.

Como parte da metodologia desenvolvida, está a abordagem de auto-configuração ou configuração automática de rede neural emulando algum método de assimilação de dados. Ou seja, a identificação da melhor topologia de uma rede neural, que foi tratada como um problema de otimização, em que cada ponto no espaço de busca representa uma arquitetura com diferentes parâmetros e pesos.

A meta-heurística MPCA foi empregada para otimizar os seguintes parâmetros: número de camadas ocultas ($n_{\text{camadas\_ocultas}}$), número de neurônio em cada camada oculta ($n_{\text{neuronios}}$), taxa de aprendizagem (η), *momentum* (α) e o tipo de função de ativação ($f_{ativação}$) empregada. Os valores permitidos para estes parâmetros são apresentados na Tabela 6.1.

Tabela 6.1 - Parâmetros que definem as arquiteturas de RNAs.

Tipo	Parâmetros	Valores
Discreto	$n_{camadas\_ocultas}$	$[1 \dots 3]$
	$n_{neurônios}$	$[1 \dots 32]$
	$\{f_{ativação}\}$	{tanh (1)}
		{sigmoide (2)}
		{gaussiana (3)}
Contínuo	η	$[0,0 \dots 1,0]$
	α	$[0,1 \dots 0,9]$

A assimilação de dados pode ser expressa como:

$$x_n^a = F_{RNA}(x_n^{\text{Obs}}, x_n^f) \tag{6.1}$$

em que RNA representa uma perceptron de múltiplas camadas. Essa rede neural requer treinamento supervisionado, sendo que saída desejada é a estimativa de análise com o filtro de Kalman.

Para a realização da assimilação, o problema de otimização utilizou o MPCA na determinação da configuração de RNA, uma abordagem estocástica, portanto vários experimentos serão executados, a fim de encontrar um número suficiente de soluções

representativas. Para o funcionamento do MPCA, foram definidos um conjunto de parâmetros, Tabela 6.2, utilizados na realização dos experimentos numéricos.

Tabela 6.2 - Parâmetros de controle e critério de parada: MPCA.

Algoritmo	Parâmetros	Valores
MPCA	*Nexperimentos*	15
	NAFO	4000
	Nprocessadores	8
	Nparticulas	8
	Nblackboard	100
	LI	$0,8$
	LS	$1,2$

Explora-se também a computação baseada em *software* e em *hardware* por meio do uso de uma RNA treinada para emular o filtro de Kalman aplicado em assimilação de dados. Um dos objetivos é fazer uma comparação numérica dos resultados e fazer um estudo sobre o desempenho da versão híbrida em relação a uma versão completamente em *software*.

Para todos os testes, utilizou-se o sistema Cray XD1. As execuções em *software* ocorreram em uma CPU do *blade* (AMD Opteron 64 bits – 2,4GHz). A implementação da RNA em FPGA foi executada em FPGA (Virtex II Pro XC2VP50 a 190MHz). As próximas seções descrevem os detalhes desses testes e apresentam os resultados e, por fim, algumas discussões

6.1 Assimilação: equação da Onda 1D – *Software* e *Hardware*

A equação da Onda 1D é é usada para simular uma dinâmica oceânica muito simplificada (BENNETT, 2002). O modelo foi integrado com o método de *Crank Nicholson* (LYNCH, 2004) com derivada espacial de quarta ordem. As observações sintéticas foram geradas a partir da integração do modelo com adição de um ruído aleatório de variância $0,5$. A curva que representa a "verdade" para o método de assimilação foi obtida baseado em na integração da equação da onda sem ruído, ou seja, quanto mais próximos as estimativas obtidas com a RNA emulando o filtro de Kalman estiver da dinâmica verdadeira do sistema, melhor será a estimativa.

Na implementação do filtro de Kalman, os seguintes valores são utilizados: para a discretização espacial usou-se $N_x = 128$ pontos; $Q_t = 0,1I$ – matriz da covariância do erro de modelagem; $R_t = 0,5$ – matriz de covariância do erro de observação;

$H = I$ – operador que representa o sistema de observação (FURTADO, 2012). A matriz de covariância do erro de previsão foi inicializada por

$$P_0^f = \begin{cases} 10(x_0^f)_i^2 & \text{para } i = j \ , \\ 0 & \text{para } i \neq j \ . \end{cases} \tag{6.2}$$

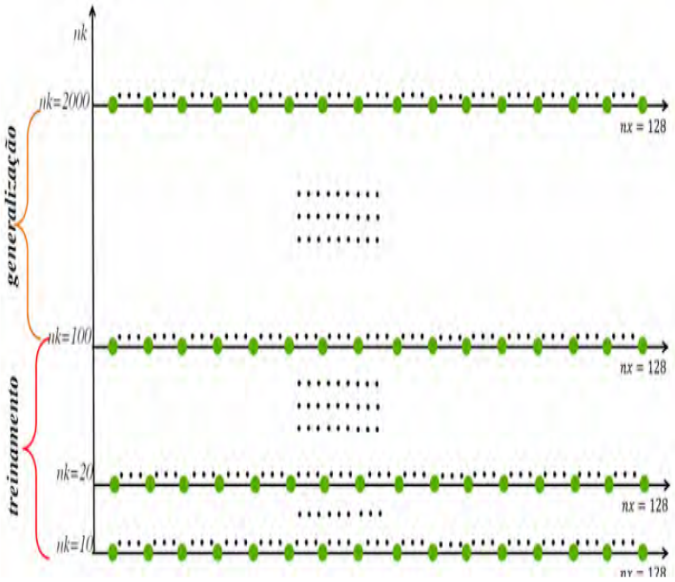
O processo de assimilação é realizado a cada 10 passo de tempo.

A estratégia de definição automática da rede utilizando a meta-heurísticas MPCA, elimina do especialista a responsabilidade de definir uma arquitetura ótima para a rede neural. Neste experimento, o processo de otimização tem início com uma arquitetura mínima de uma RNA válida e durante o processo de otimização, os parâmetros são alterados. As melhores arquiteturas encontradas com o MPCA correspondem aos parâmetros apresentados na Tabela 6.3, para efeito de comparação, foram avaliados o erro quadrático médio (EQM) do conjunto de teste.

O funcionamento desta arquitetura de rede é realizado por meio de dois passos que são: o treinamento e a ativação. Na fase de treinamento, apresentam-se um conjunto de dados: dados de entrada e dados de saída desejada. Neste caso, os dados usados como saída desejada é formado pela estimativa obtida com o filtro de Kalman, método de assimilação de dados. A Figura 6.1 descreve o modo como os dados de treinamento e generalização da rede foram selecionados. O modelo foi integrado em 2000 passos de tempo com 128 pontos na discretização espacial. Os dados integrados até o passo de tempo $n_k = 100$ formaram o conjunto de treinamento. As observações foram inseridas a cada 10 passos de tempo, representadas pelos pontos verdes. Os dados de $n_k = 101$ até $n_k = 2000$ foram usados para a generalização da rede.

Os pesos sinápticos são obtidos após a fase de treinamento. Em seguida, os neurônios da rede são ativados com novos dados de entrada que não foram usados no treinamento verificando a capacidade de generalização da informação de uma rede neural artificial. Na fase de treinamento de uma RNA, há a atualização dos pesos pelo algoritmo *back-propagation*. Na fase de ativação, os pesos sinápticos são fixos, que foram determinados na fase de treinamento.

Figura 6.1 - Conjunto de dados de treinamento – Onda 1D



A arquitetura, definida empiricamente, é chamada por RNA-Empírica (FURTADO, 2012). As arquiteturas da RNA-1 e RNA-2 foram definidas pelo MPCA, mas o experimento numérico foi realizado com duas suposições iniciais diferentes para pesos e viés. Na primeira experiência (RNA-1), a estimativa inicial foi o valor $= 0,5$ para todos os pesos e viés. Para o segundo experimento (RNA-2), os pesos e bias foram inicializados com valores aleatórios.

Tabela 6.3 - Arquiteturas de RNAs para Onda1D.

Parâmetros	RNA-Empirica	RNA-1	RNA-2
$n_{camadas\_ocultas}$	1	1	1
$n_{neurônios}$	3	23	1
Taxa aprendizado (η)	$0,53$	$0,4$	$0,9$
Momentum (α)	$0,20$	$0,6$	$0,0$
$f_{ativação}$	1	1	1

Na Figura 6.2, a curva azul corresponde ao estado real, a referência de verdade. Considera-se como verdade a integração do modelo sem ruído. A curva vermelha representa a análise, estimado pelo filtro de Kalman e a observação é representada pelos quadrados verdes.

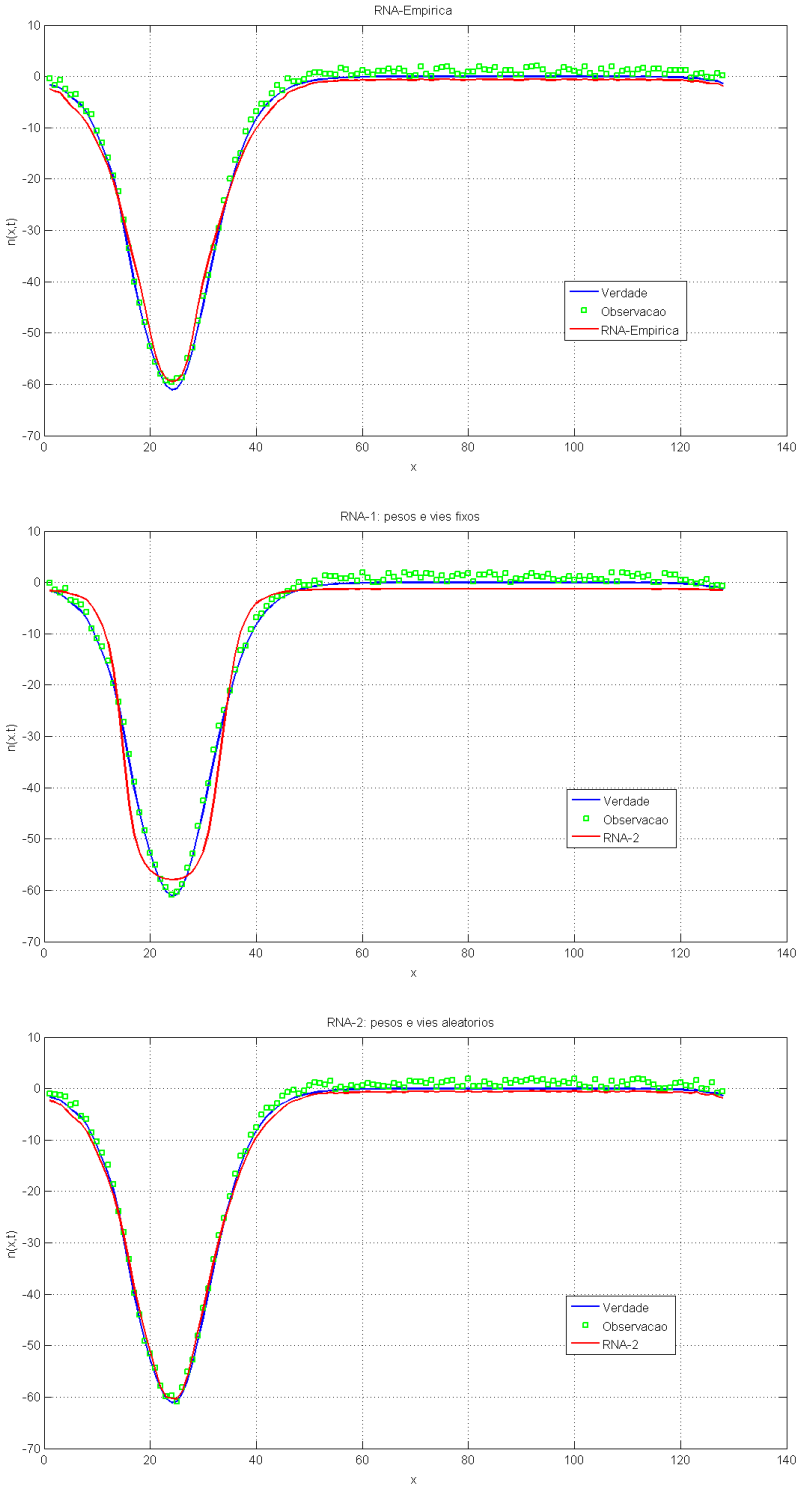
Ao observar os resultados obtidos com a assimilação, verifica-se que a RNA-Empírica obteve um bom resultado comparado com a RNA-1, cuja arquitetura foi definida pelo MPCA. Porém o melhor resultado foi obtido pela RNA-2, praticamente reproduziu a análise com grande fidelidade a dinâmica real. A RNA-2 foi definida pelo MPCA e teve seus pesos e viés inicializados aleatoriamente, o resultado numérico demonstra uma dependência do cálculo dos pesos e viés como função da estimativa inicial para os pesos. Esta discrepância nos resultados não está ligada a meta-heurística utilizada na configuração da rede neural, mas sim, com o algoritmo de treinamento da rede, regra delta generalizada (WILLIAMS; HINTON, 1986). O método da regra delta utiliza gradiente descendente do erro da rede para adaptação dos pesos, contudo possui convergência lenta e facilmente pode ficar preso em mínimos locais.

Uma maneira de tratar a dependência da RNA da forma como os pesos e viés são inicializados seria por meio do uso de métodos de busca globais de otimização, o ajuste de pesos sinápticos pode ser formulado como um problema de otimização, em que cada solução mantém um conjunto de valores para os pesos sinápticos das conexões. O MPCA foi utilizado como algoritmo de treinamento efetuando este ajuste durante o treinamento, uma nova estratégia de aprendizagem. Essa metodologia obteve ótimos resultados conforme pode ser visto em Anochi et al. (2003).

Os dois melhores resultados obtidos para assimilação de dados da equação da Onda 1D foram implementadas em FPGA. A RNA-Empírica descrito por (FURTADO, 2011) com duas entradas (previsão e observação), três neurônios na camada oculta e um neurônio na camada de saída (análise) e a RNA-2 que corresponde a melhor topologia definida pelo MPCA para esta aplicação. O projeto foi implementado para o sistema híbrido *Cray-XD1* e a implementação em VHDL foi compilada. Para utilzar menos memória e acelerar a computação na FPGA, não foram efetuadas operações de ponto flutuante: os cálculos foram feitos usando 16 bits ponto fixo, sendo divididos pela metade para as partes inteiras e fracionárias. A função de ativação é computada e armazenada em uma unidade *Lookup Tables* (LUT).

Para avaliar os resultados das implementações do neuro-computador com FPGA, foi utilizada uma versão totalmente em *software* (CPU: Unidade Central de Processamento). Os resultados são mostrados nas figuras 6.3 e 6.4 , correspondendo as implementações em *software* e *hardware*, e a diferença quadrática entre estas duas implementações. Os resultados para a RNA definida por um especialista são mostrados na Figura 6.3, e os resultados para RNA autoconfigurada pelo MPCA estão representados na Figura 6.4. Os valores mostrados na Tabela 6.4 correspondem a

Figura 6.2 - (a) RNA-Empírica; (b)RNA-1: MPCA, pesos e viés fixos; (c) RNA-2: MPCA, pesos e viés aleatórios



erro quadrático médio e variância.

Figura 6.3 - RNA-Empírica (superior: implementação em *software*, meio: implementação em *hardware*, inferior: diferença quadrática).

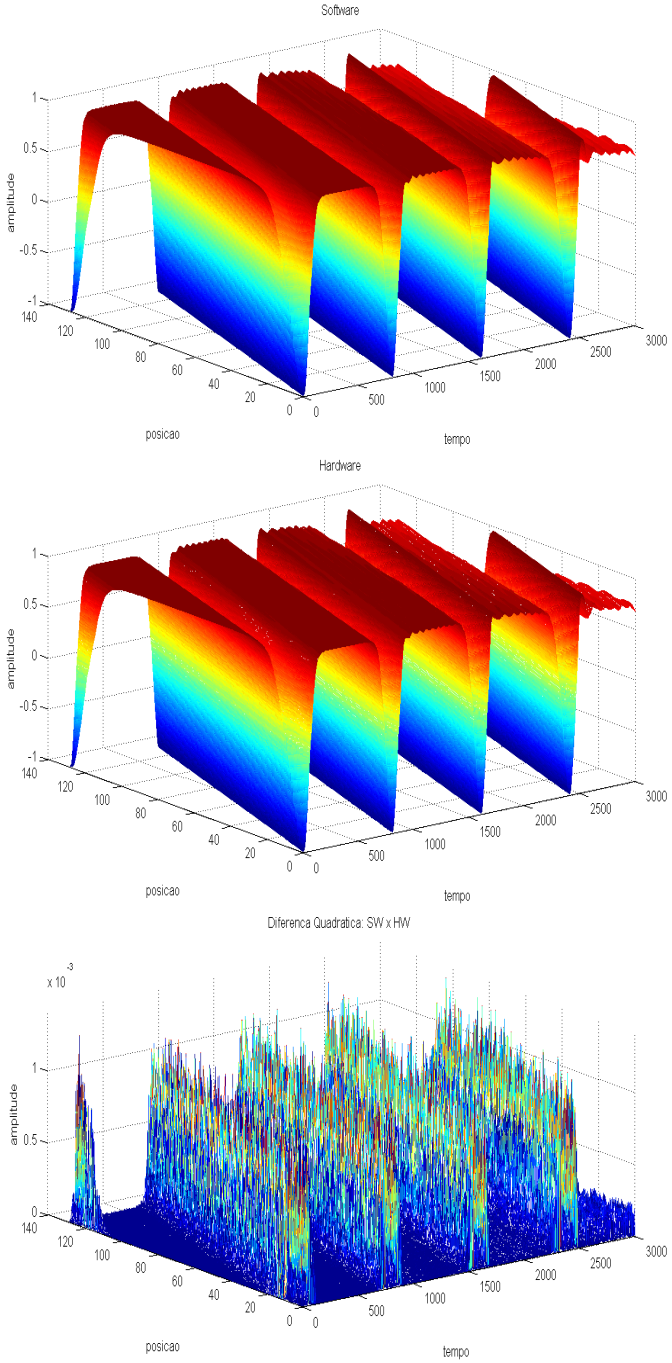


Figura 6.4 - RNA-2 (superior: implementação em *software*, meio: implementação em *hardware*, inferior: diferença quadrática).

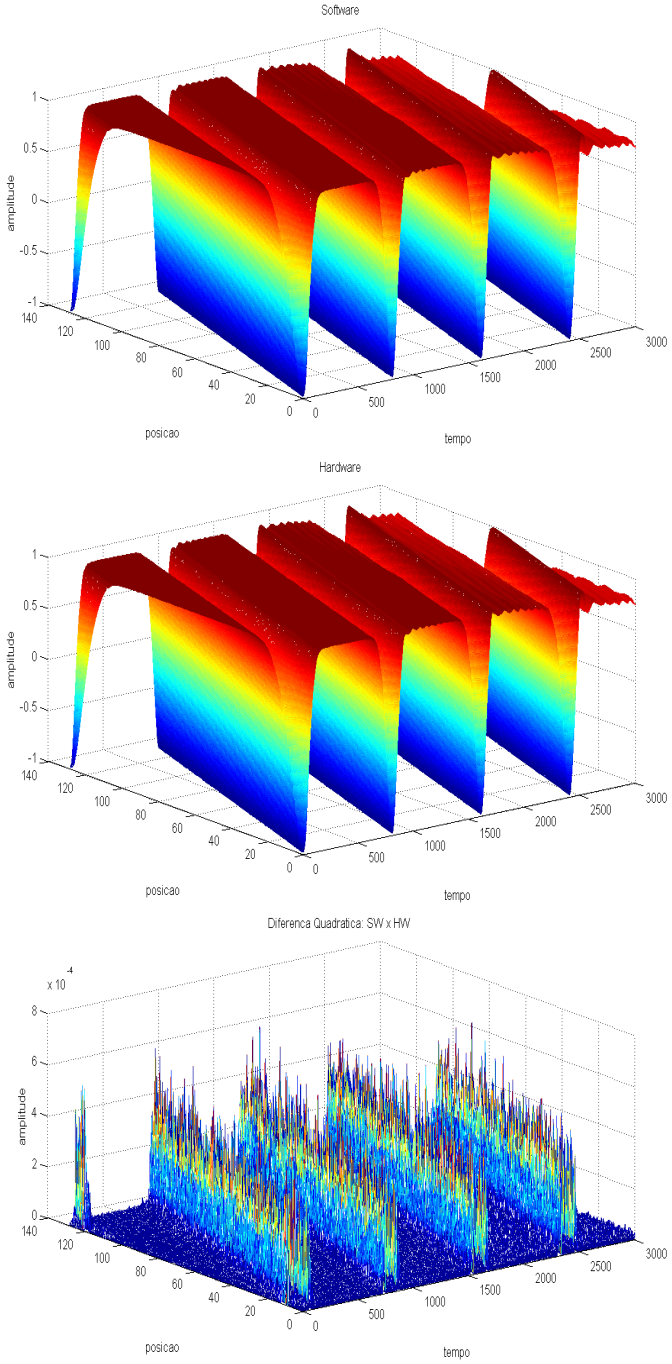


Tabela 6.4 - Erros das redes RNA-Empírica e RNA-2 em assimilação de dados para Onda-1D.

	RNA-Empírica	RNA-2
Erro quadrático médio	$1,68 \times 10^{-5}$	$5,18 \times 10^{-5}$
Variância	$1,69 \times 10^{-9}$	$1,79 \times 10^{-8}$

As redes neurais artificiais podem ser empregadas como um método de assimilação de dados. Aqui, a RNA perceptron de múltiplas camadas foi aplicada para emular o filtro de Kalman para a Onda 1D, um sistema dinâmico. A implementação em FPGA funcionou bem, em que a aritmética de ponto fixo foi adotada para evitar restrições de memória.

Na implementação em FPGA, a função de ativação não é codificada como uma função matemática, foi utilizada uma abordagem com *lookup table*. A estratégia para a configuração automática da RNA usando meta-heurística MPCA foi efetiva, com aplicação para assimilação de dados. De fato, a arquitetura definida pelo MPCA para a RNA produziu melhores resultados do que a configuração definida por um especialista.

6.2 Assimilação: sistema de água rasa 2D – *Software* e *Hardware*

O modelo de água rasa 2D foi discretizado espacialmente com grade-C de Arakawa e um esquema *forward-backward* para integração numérica no tempo (MESINGER; ARAKAWA, 1976). A Tabela 6.5 mostra os parâmetros utilizados para a integração do modelo, estes parâmetros foram adotados para reproduzir o experimento descrito no livro de Bennett (2002).

Tabela 6.5 - Parâmetros do modelo: sistema de água rasa 2D.

Parâametros	Valores
H	5000 m
T	$1,8 \times 10^4$
g	$9,806\ \mathrm{m\,s^{-2}}$
f	$1,01,0^{-4}\ \mathrm{s^{-1}}$
r_u	$(1,8 \times 10^4 s)^{-1}$
r_v	$(1,8 \times 10^4 s)^{-1}$
r_q	$(1,8 \times 10^4 s)^{-1}$
C_d	$1,6 \times 10^{-3}$
ρ_a	$1,275\ \ \mathrm{kg\,m^{-3}}$
ρ_w	$1,0$ x 10^3 kg $\mathrm{m^{-3}}$

Como já mencionado, o MPCA foi aplicado para otimizar os parâmetros de RNA e a Tabela 6.6 mostra os resultados obtidos, correspondendo a uma média de 15 experimentos com sementes gerando diferentes números aleatórios.

Os parâmetros utilizados para executar o MPCA foram: 8 partículas, 8 processadores, 100 iterações, com ciclo de comunicações a cada 10 iterações, 10 iterações de busca local (esquema usado para explorar a melhor solução em torno do novo local de partículas), LI = 0,8 e LS = 1,2. O critério de parada utilizado foi o número máximo de avaliações da função objetivo (para o exemplo trabalhado: 500).

Tabela 6.6 - Topologias para RNA.

Parâmetros	RNA-Empirica	RNA-MPCA
Camada Oculta	1	1
Neurônios camada oculta	10	10
função de ativação	tanh	tanh
Taxa aprendizado (η)	0,70	0,94
Momentum (α)	0,00	0,00
Erro Quadrático	$0,5264$	$0,1583$

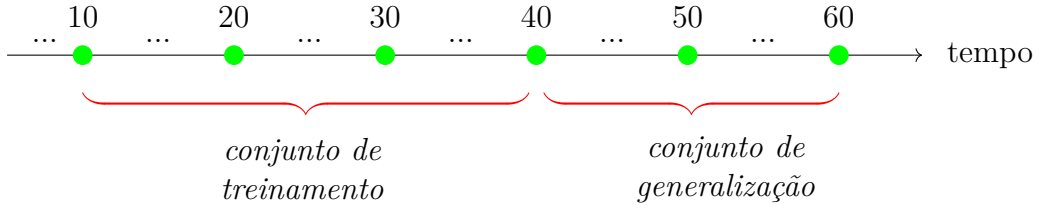
As equações de água rasa foram integradas em 60 passos no tempo, a variável q foi inicializada com função gaussiana e as variáveis $u = v = 0$. O processo de assimilação de dados foi feito a cada 10 passos de tempo. Tabela 6.7 mostra os parâmetros do modelo numérico usados neste experimento

Tabela 6.7 - Parâmetros para o modelo numérico.

Parâmetros	Valores
Pontos grade x	NI= 40
Pontos grade y	NJ= 40
Passos no tempo	NK= 60
Espaçamento x	Δx = 100 km
Espaçamento y	Δy = 100 km

Figura 6.5 ilustra como os dados para o treinamento de RNA foram selecionados. Para o treinamento da rede foram utilizados os dados desde o início até o passo 40 e para a generalização do passo 41 até o passo 60.

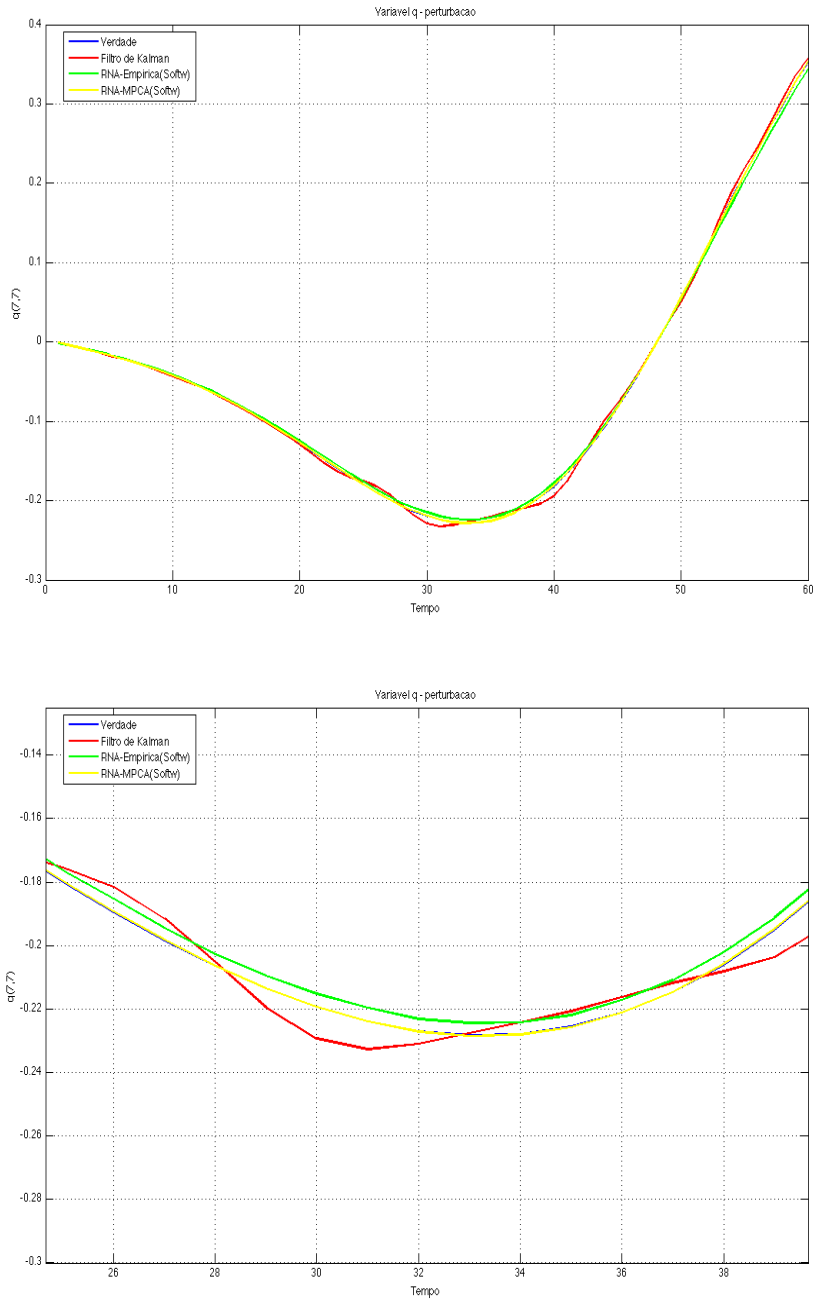
Figura 6.5 - Conjunto de dados de treinamento para o modelo água rasa 2D.



As Figuras 6.6, 6.7 e 6.8 mostram a evolução temporal no ponto (x, y) para $x = y = 7$. Um dos objetivos é identificar uma arquitetura ótima de rede neural para emular o processo de assimilação realizado pelo filtro de Kalman. A Figura 6.6 mostra os resultados obtidos com a implementação em *software* (CPU: Unidade Central de Processamento). A curva azul representa a referência verdadeira, a curva vermelha representa a estimativa do filtro de Kalman, as curvas verde e amarela representam os resultados da estimativa de RNA, a diferença entre elas está no modo pelo qual a RNA foi configurada: empiricamente (curva verde) (FURTADO, 2011), com MPCA (curva amarela). Pode-se verificar que o resultado obtido com a RNA-MPCA é melhor do que o obtido com a RNA-Empirica, e ambos os resultados são melhores do que os obtidos com o filtro de Kalman.

Outro objetivo é comparar os resultados de RNA implementados em *software* e *hardware*. A Figura 6.7 mostra a comparação entre os resultados: filtro de Kalman e RNA definida por um especialista com implementação de *software* e implementação híbrida no sistema Cray XD1. Novamente, os melhores resultados foram obtidos com

Figura 6.6 - Evolução temporal da variável q no instante $(7, 7)$ – *Software* RNA-Empírica e RNA-MPCA (figura inferior é uma visão mais detalhada).



RNA, tanto em *software* quanto em *hardware*.

A Figura 6.8 mostra os resultados obtidos com RNA-MPCA, ou seja, RNA autoconfigurada, implementada em *software* e *hardware*. Mais uma vez, RNA tem os melhores resultados.

Os resultados apresentados na Figura 6.9 correspondem à diferença quadrática para a evolução temporal no ponto (x, y) para $x = y = 49$ para a RNA autoconfigurada entre estas duas implementações: *software* e *hardware*. Pode-se notar que a implementação em *hardware* possui um erro menor, ou seja, uma melhor qualidade na resposta.

Após avaliação e validação dos resultados obtidos com a metodologia apresentada nesta tese, foram calculados o desempenho computacional usando o mesmo conjunto de dados para a execução em *software* e *hardware*. Os tempos referentes a média de 10 realizações para assimilação, são apresentados na Tabela 6.8.

Tabela 6.8 - Tempos médios de 10 realizações para assimilação 2D.

Tempo software total	121709 us
Tempo hardware total	209187 us

Verifica-se que o tempo gasto com a implementação em *hardware* é maior que o tempo gasto com a implementação em *software*. A Tabela 6.9 corresponde a uma avaliação de desempenho mais detalhada dos tempos gastos com a implementação em *hardware*, pode-se observar que o responsável pelo baixo desempenho é a comunicação, ou seja, o envio e recebimento de dados pelo FPGA. A comunicação é um dos grandes desafios nas implementações de alto desempenho. O tempo gasto, especificamente, no FPGA é ínfimo comparado com o tempo total gasto na implementação em : 2 us – 209187 us.

Tabela 6.9 - Tempos: execução em FPGA, transferência de dados.

Tempo: envio CPU–FPGA	181635 us
Tempo: computação FPGA	2 us
Tempo: envio FPGA–CPU	9455 us

Foi feito um outro teste, utilizando MPI, em que dois processadores e dois FPGAs

Figura 6.7 - Evolução temporal da variável q no instante $(7, 7)$ – *Software* e *hardware* RNA–Empírica (figura inferior é uma visão mais detalhada).

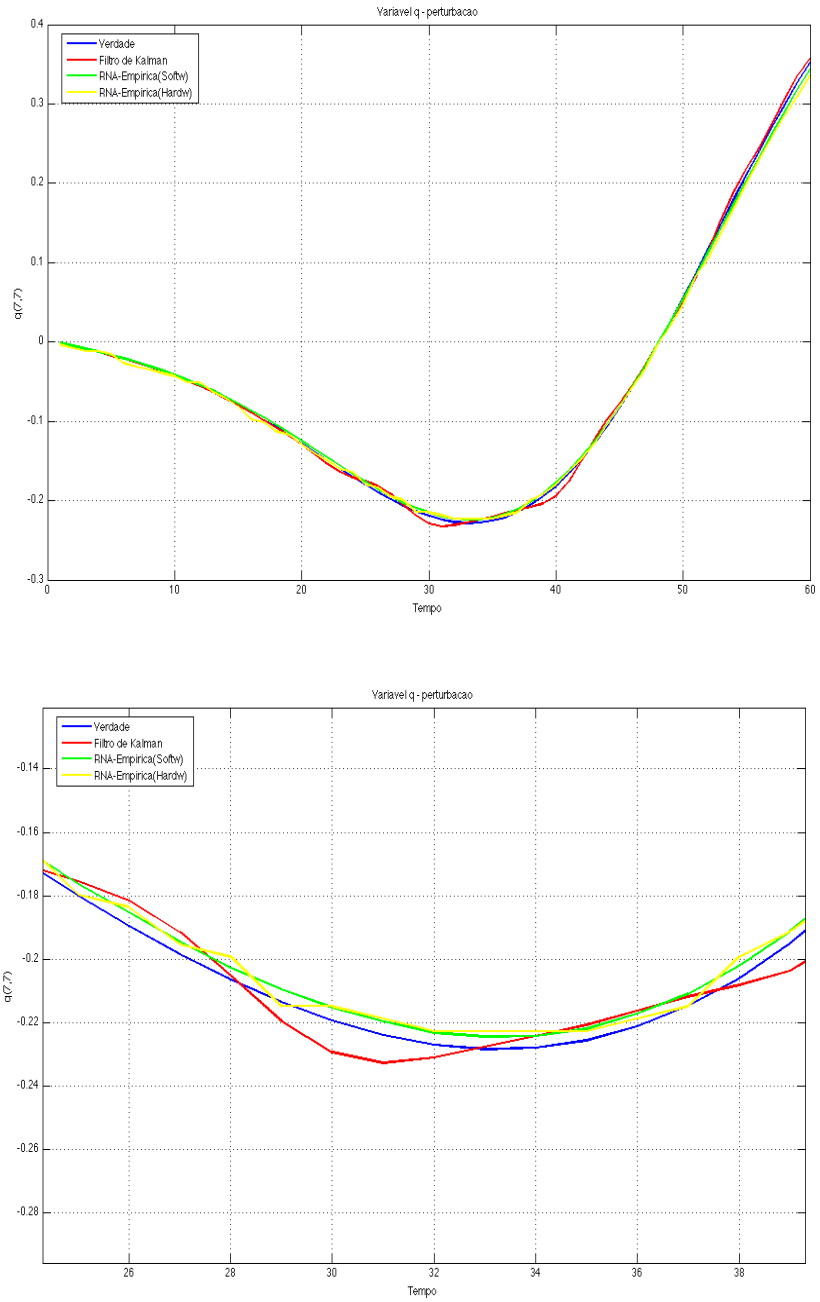


Figura 6.8 - Evolução temporal da variável q no instante $(7, 7)$ – *Software* e *hardware* RNA–MPCA (figura inferior é uma visão mais detalhada).

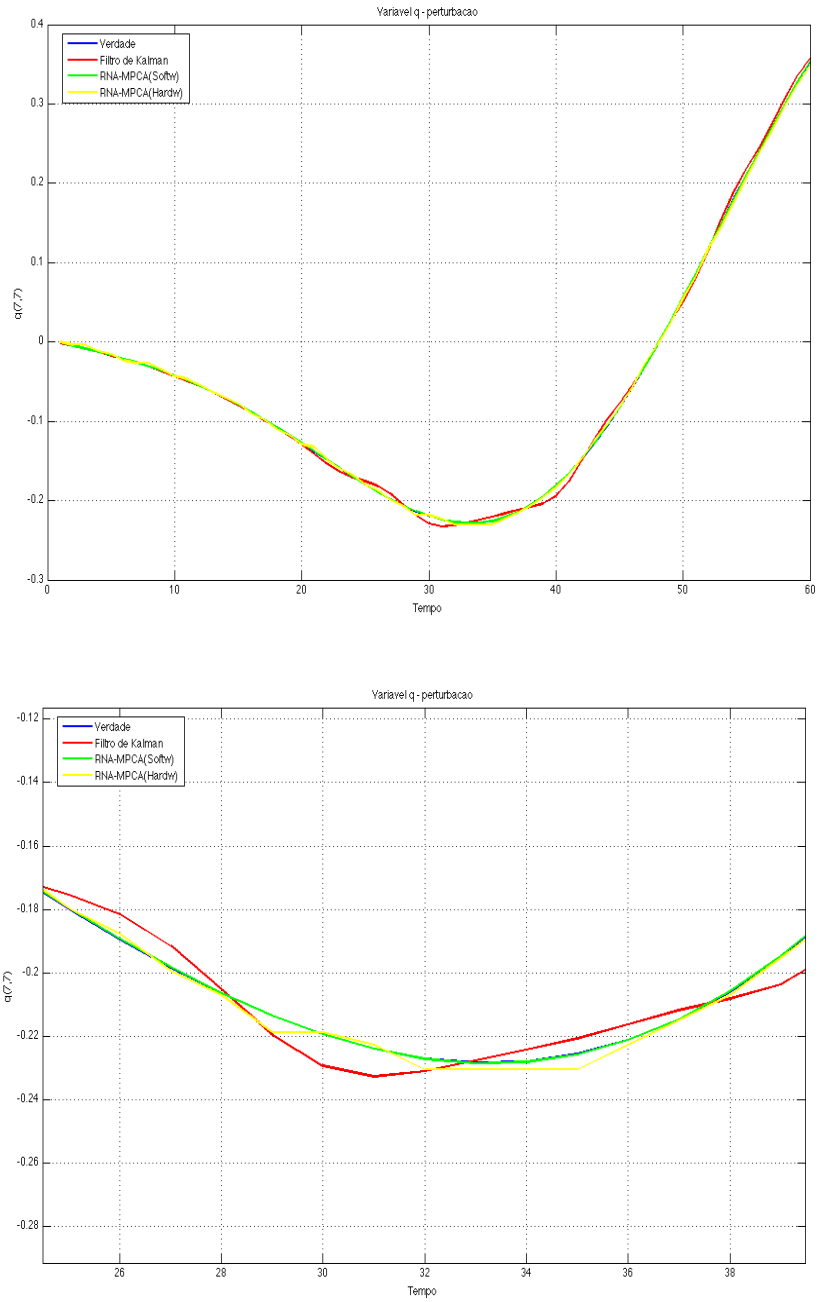
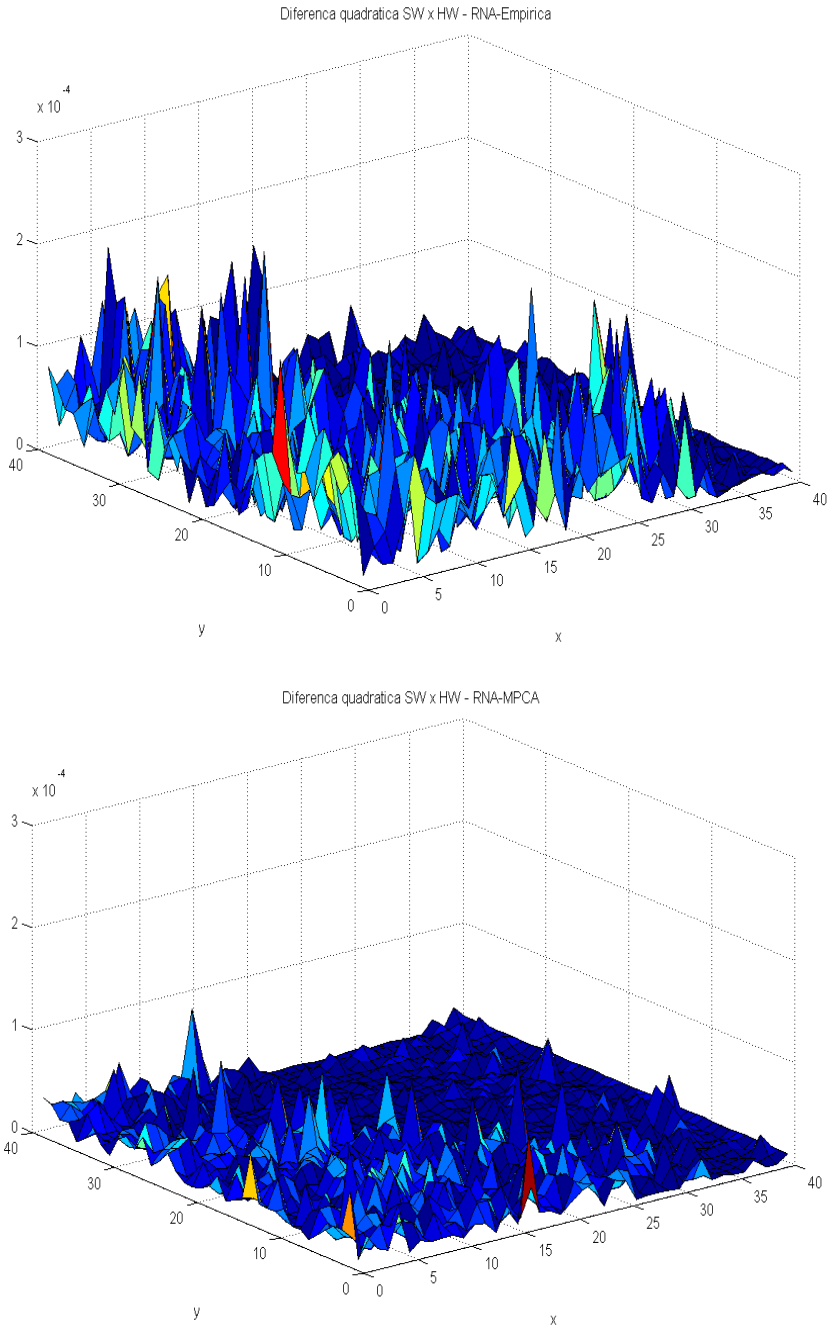


Figura 6.9 - Diferença quadrática *Software* × *Hardware* - RNA–MPCA - q - Tempo = 59



foram empregados na execução de apenas 1 ciclo de assimilação, a Tabela 6.10 apresenta os tempos gastos em ambos processadores, como pode se esperar, a diferença é mínima.

Tabela 6.10 - Tempos de 1 ciclo de assimilação com RNA: 2 processadores.

Tempo hardware total (Proc-01)	2955 us
Tempo hardware total (Proc-02)	3130 us

7 CONCLUSÕES

Um dos desafios do mundo moderno é lidar com grande massa de dados de tipos distintos, distribuídos em banco de dados estruturados de forma diversa e usando plataformas de *software* diferentes. Extrair informação neste cenário é a tarefa da área de *Big Data*. Gerar conhecimento de uma massa gigantesca de dados em aplicações científicas é o que se chama de *Data Science*.

Há duas tendencias claras na área de computação. Uma delas foi mencionada no parágrafo anterior: *Data Science* e *Big Data*. Outra é o uso cada vez mais disseminado de *computação híbrida*, isto é, arquiteturas heterogêneas de processamento.

O presente estudo esteve voltado à aplicação de redes neurais em assimilação de dados. Este algorítimo permite reduzir o esforço computacional para a tarefa essencial de inserção de dados observacionais na previsão operacional com uso de sistemas dinâmicos. O cálculo da *análise* tornou-se o módulo de maior demanda computacional dos modelos de previsão numérica do tempo.

De fato, como mostrado por Cintra (2010), a rede neural usada para emular o *LETKF* para o modelo *SPEEDY* (código espectral meteorológico 3D global), durante simulação de um mês, foi capaz de reduzir o cálculo computacional para a assimilação de dados de 4 horas e 20 minutos para *menos* de 3 min (redução de mais de 98% do tempo de CPU). Na simulação de **??**), 6 RNAs foram definidas para diferentes regiões do globo. O modelo SPEEDY é um código computacional simplificado e de baixa resolução. Para aplicação em modelos operacionais de previsão de tempo, o globo deve ser dividido em mais regiões (CINTRA; VELHO, 2014). Isso indica a necessidade de um processo automático para configurar as RNAs.

Um dos focos da tese foi o desenvolvimento e aplicação de uma metodologia para a configuração automática de redes neurais supervisionadas. O processo é formulado como um problema de otimização, onde a função objetivo, Equação 3.7, tem duas componentes: um termo de quantificação de complexidade da RNA e um termo que quantifica o erro da RNA em emular o processo de assimilação de dados pelo filtro de Kalman. O problema de otimização é solucionado pela meta-heurística MPCA – ver seção 3.3. A análise calculada pelo filtro de Kalman é usada para gerar conjuntos de treinamento e de verificação de teste de genralização, ambos usados na função custo para configurar a RNA ótima aplicada à assimilação de dados. Na presente pesquisa, a rede *feed-forward* perceptron de múltiplas camadas foi empregada.

Outro objetivo da tese foi implementar a rede neural para assimilação de dados em FPGA, numa arquitetura onde CPU está associada ao co-processador reconfigurável. O sistema usado foi o computador multi-processado Cray XD1, onde 6 nós de processamento, onde cada *blade* tem 2 processadores AMD associado por uma FPGA Xilinx Virtex II Pro. Todos os componentes no computador Cray XD1 estão interligados pela rede *Rapidarrray*.

Modelos simplificados de dinâmica do oceano, equação da Onda 1D e equações de água rasa 2D, foram utilizados para testar as metodologias propostas. Conforme mostrado no Capítulo 6, as RNAs configuradas pelo MPCA, tanto para o modelo de Onda 1D (seção 6.1) como para o sistema de água rasa 2D (seção 6.2), foram capazes de emular a *análise* computada pelo filtro de Kalman. Mais ainda, as redes auto-configuradas obtiveram um desempenho superior a das redes configuradas por especialistas. Mostrando, mais uma vez, que rede neural é um método competitivo para assimilação de dados.

O uso de FPGA em computação é crescente e atualmente existem empresas voltadas a desenvolver computadores de uso específicos baseados exclusivamente em FPGA (http://www.copacobana.org/).

A implementação de RNAs em FPGA para executar a tarefa de assimilação de dados foi efetiva, pois a diferença entre a análise calculada por *software* e por *hardware* foi muito pequena. Assim, o processamento da assimilação por FPGA é uma alternativa confiável e aplicável ao processo de assimilação de dados.

Em sistemas computacionais de grande porte, uma preocupação cada vez mais relevante é a demanda de energia requerida por estes sistemas. Por exemplo, o projeto *Blue Waters*, do National Center for Supercomputer Applications (NCSA)[1] da Universidade de Illinois em Urbana Champaign (EUA), é um sistema híbrido CPU-GPU e opera com 13,5 PFlops como pico de processamento – na verdade, este sistema foi projetado para operar com 1 PFlop de processamento sustentado. Para operar, o sistema *Blue Waters* poderá consumir até 24 MW, mas a demanda de energia regular é assumida em 15 MW. Porém, como citado, esta é uma máquina projetada para superar o desafio de 1 PFlop sustentado. O desafio atual está em superar a marca de exaflop. Considerando um cálculo linear de demanda de energia, uma máquina exaflop com a tecnologia *Blue Waters* iria consumir perto de 1800 MW!

Em um artigo recente, pesquisadores chineses realizaram uma avaliação interessante

[1] www.ncsa.illinois.edu/enabling/bluewaters .

Tabela 7.1 - Equação de água rasa em sistemas heterogêneos, segundo (FU H ; GAN, 2017).

Sistema	Desempenho (GFlops)	*speed-up*	consumo (W)
CPU 6-cores	12	1×	—
CPU 12-cores	24	2×	—
GPU Fermi	140	12×	360
MIC Xeon Phi	200	17×	815
FPGA Virtex-6	900	75×	514

sobre processamento em arquiteturas heterogêneas: CPU+GPU (supercomputador Tianhe-1A), CPU+MIC (supercomputador Tianhe-2), CPU+FPGA (arquitetura Maxeler DFE) (FU H ; GAN, 2017). A aplicação da pesquisa dos cientistas chineses foi o uso de equações de água rasa 2D, onde as equações foram discretizadas sobre uma malha *cubed-sphere*. Os resultados obtidos mostram que o processamento com co-processadores obteve *speed-up* superior, sendo que o melhor desempenho foi obtido com FPGA. O resultado de melhor desempenho não ficou vinculado na redução do esforço computacional. A arquitetura híbrida CPU+FPGA foi mais de 12 vezes mais eficiente em consumo de energia (GFlops/Watt) do que o sistema Tianhe-2 (CPU+MIC) – o ambiente com CPU+GPU (Tianhe-1) foi 1,2 vezes mais eficiente que a arquitetura CPU+MIC. A tabela 7 resume o resultado dos pesquisadores chineses – a coluna do consumo de energia mostra o consumo por nó de processamento.

7.1 Trabalhos Futuros

A pesquisa em configuração automática de redes neurais está no início. Outras topologias de redes já foram usadas com sucesso no processo de assimilação de dados (HÄRTER, 2004; HARTER; Campos Velho, 2008). Particularmente interessante é o uso de redes recorrentes e de redes não supervisionadas.

A aplicação da rede MLP (ou PMC) em modelos globais 3D próximos aos códigos usados em centros operacionais tem sido pesquisado por Cintra e colaboradores (CINTRA; Campos Velho, 2012; CINTRA; VELHO, 2014). O uso da metodologia de redes auto-configurada nestes contextos é imediata.

Em futuro imediato, pretende-se realizar um estudo comparativo entre diferentes tipos de FPGA. O *cluster* Lacibrido do Laboratório Associado de Computação e Matemática Aplicada (LAC) do INPE tem nós de processamento híbrido com CPU Intel de 10-cores e 12-cores, CPU ARM de 4-cores e 8-cores, GPUs Nvidia K20 e K40, FPGAs Xilinx Virtex-6 e Virtex-7 e MIC Intel Xeon Phi de 60-cores. A pes-

quisa visará primeiramente implementação e medidas com FPGAs, onde pretende-se avaliar o tempo de processamento, velocidade de transferência de dados entre CPU e FPGA, impacto do uso de operações de ponto flutuante versus ponto fixo, operações com precisão dupla e precisão mista.

A avaliação do consumo de energia é um tema muito relevante. Visto que sistemas de supercomputação petaflops (e superiores) tem necessidade de enorme demanda de energia. Hoje já existe um *ranking* para eficiência energética de supercomputadores: *The Green 500*[2]. Desta forma, pretende-se também avaliar a demanda de energia de cada co-processador utilizado, com foco na tendência recente de *green computing*

[2] https://www.top500.org/green500/ .

REFERÊNCIAS BIBLIOGRÁFICAS

ANOCHI, J. A.; SAMBATTI, S. B.; LUZ, E. F. P. d.; Campos Velho, H. F. d. New learning strategy for supervised neural network: Mpca meta-heuristic approach. In: CONGRESSO BRASILEIRO DE INTELIGÊNCIA COMPUTACIONAL, 21., 2013, Recife Natal (RN), Brasil. **Anais...** Recife: Sociedade Brasileira de Inteligência Computacional, 2003. p. 01–06. 29, 65

ANTONIOU, A.; LU, W.-S. **Practical optimization**: algorithms and engineering applications. [S.l.]: Springer Science & Business Media, 2007. 25

BECCENERI, J. C. **Meta-heurísticas e otimização combinatória:**: Aplicações em problemas ambientais. São José dos Campos: [s.n.], 2008. 25, 27

BENARDOS, P.; VOSNIAKOS, G.-C. Optimizing feedforward artificial neural network architecture. **Engineering Applications of Artificial Intelligence**, Elsevier, v. 20, n. 3, p. 365–382, 2007. 24

BENNETT, A. F. **Inverse modeling of the ocean and atmosphere**. [S.l.]: Cambridge University Press, 2002. 3, 53, 57, 58, 59, 62, 69

BORGES, R. R.; IAROSZ, K. C.; BATISTA, A. M.; CALDAS, I. L.; BORGES, F. S.; LAMEU, E. L. Sincronização de disparos em redes neuronais com plasticidade sináptica. **Revista Brasileira de Ensino de Física**, scielo, v. 37, p. 2310–1 – 2310–9, 06 2015. ISSN 1806-1117. Disponível em: <http://www.scielo.br/scielo.php?script=sci_arttext&pid=S1806-11172015000200011&nrm=iso>. 15

BOUTTIER, F.; COURTIER, P. **Data assimilation concepts and methods March 1999**. [S.l.: s.n.], 2002. 59 p. 5, 6, 7

BRAGA, A. P.; CARVALHO, A. P. L. F.; LUDERMIR, T. B. **Redes Neurais Artificiais:**: Teoria e aplicações. [S.l.]: LTC, 2000. 16, 17, 18, 19, 20, 21, 23, 24

BRODTKORB, A. R.; DYKEN, C.; HAGEN, T. R.; HJELMERVIK, J. M.; STORAASLI, O. O. State-of-the-art in heterogeneous computing. **Scientific Programming**, v. 18, p. 1 – 33, January 2010. 42

CARDOSO, J. M.; DINIZ, P. C.; WEINHARDT, M. Compiling for reconfigurable computing: A survey. **ACM Computing Surveys (CSUR)**, ACM, v. 42, n. 4,

p. 13, 2010. 44, 45

CARDOSO, J. M. P.; DINIZ, P. C.; WEINHARDT, M. Compiling for reconfigurable computing:: A survey. **ACM Comput. Surv.**, v. 42, n. 4, 2010. 44

CARVALHO, A. R. **Uso de redes neurais otimizadas para recuperaçâo do perfil de concentração de gases traço atmosféricos a partir de dados de satélites**. 208 p. Tese (Doutorado) — Instituto Nacional de Pesquisas Espaciais, São José dos Campos, 2011-06-27 2011. Disponível em: <http://urlib.net/sid.inpe.br/mtc-m19/2011/06.06.12.20>. Acesso em: 28 jul. 2017. 26

CARVALHO, A. R. **Uso de redes neurais para recuperação do perfil de concentração de gases traço atmosféricos a partir de dados de satélites**. Tese (Tese de Doutorado) — Instituto Nacional de Pesquisas Espaciais (INPE), São José dos Campos, Junho 2011. 2, 25, 26

CHAMBERLAIN, R. D.; LANCASTER, J.; CYTRON, R. K. Visions for application development on hybrid computing systems. In: **Reconfigurable Systems Summer Institute**. [S.l.: s.n.], 2007. 38, 41

CHUA, B. S.; BENNETT, A. F. An inverse ocean modeling system. **Ocean Modelling**, Elsevier, v. 3, n. 3, p. 137–165, 2001. 53

CINTRA, R.; Campos Velho, H. F. de. Global data assimilation using artificial neural networks in speedy mode. In: INTERNATIONAL SYMPOSIUM UNCERTAINTY QUANTIFICATION AND STOCHASTIC MODELING, 5., 2012, Maresias, SP. **Proceedings...** Brazil, 2012. 81

CINTRA, R. S. C. **Assimilação de dados com redes neurais artificiais em modelo de circulação geral da atmosfera**. Tese (Tese de Doutorado) — Instituto Nacional de Pesquisas Espaciais, São José dos Campos, 2010-09S 2010. 1, 79

CINTRA, R. S. C.; Campos Velho, H. F. de. **Global data assimilation by artificial neural networks for an atmospheric general circulation model:**: Conventional observation. [S.l.], 2011. 13

CINTRA, R. S. C.; VELHO, H. F. de C. Data assimilation by artificial neural networks for an atmospheric general circulation model: Conventional observation.

CoRR, abs/1407.4360, 2014. Disponível em: <http://arxiv.org/abs/1407.4360>. 79, 81

COMPTON, K.; HAUCK, S. Reconfigurable computing: a survey of systems and software. **ACM Computing Surveys (csuR)**, ACM, v. 34, n. 2, p. 171–210, 2002. 39

COURANT, R.; FRIEDRICHS, K.; LEWY, H. ̈ber die partiellen differenzengleichungen der mathematischen physik. **Mathematische Annalen**, v. 100, n. 1, p. 32–74, 1928. 53

CRAY XD1 Datasheet. USA: Mendota Heights, 2005. 45, 47, 48, 49

FU H ; GAN, L. . Y. C. . X. W. . W. L. . W. X. . H. X. . Y. G. Solving global shallow water equations on heterogeneous supercomputers. **PLOS ONE**, Public Library of Science, v. 12, n. 3, p. 1–27, 03 2017. xvii, 3, 81

FURTADO, H. C. M. **Redes neurais e diferentes métodos de assimilação de dados em dinâmica não linear**. 125 p. Dissertao (Mestrado) — Instituto Nacional de Pesquisas Espaciais, São José dos Campos, 2008-02-22 2008. Disponível em: <http://urlib.net/sid.inpe.br/mtc-m17@80/2008/02.07.10.49>. Acesso em: 28 jul. 2017. 1, 6

FURTADO, H. C. M. Assimilação de dados com redes neurais artificiais em equações diferenciais. **Conferência Brasileira de Dinâmica, Controle e Aplicações**, p. 595 – 598, 2011. 12, 65, 71

FURTADO, H. C. M. **Redes neurais para assimilação de dados em um modelo de circulação oceânica**. 173 p. Tese (Doutorado) — Instituto Nacional de Pesquisas Espaciais (INPE), São José dos Campos, 2012-12-07 2012. Disponível em: <http://urlib.net/sid.inpe.br/mtc-m19/2012/11.19.16.34>. Acesso em: 28 jul. 2017. xvii, 13, 54, 59, 63, 64

GHIL, M.; MALANOTTE-RIZZOLI, P. Data assimilation in meteorology and oceanography. **Advances in geophysics**, Elsevier, v. 33, p. 141–266, 1991. 8

GOMES, V. C.; SHIGUEMORI, E. H.; CHARÃO, A. S.; VELHO, H. F. d. C. Rede perceptron de múltiplas camadas para sistema híbrido reconfigurável. In: WORKSHOP DOS CURSOS DE COMPUTAÇÃO APLICADA DO INPE, 11.

(WORCAP)., 11., 2011, São José dos Campos, SP. **Anais...** São José dos Campos: Instituto Nacional de Pesquisas Espaciais (INPE), 2011. 45, 46, 52

GOMES, V. C. F. **Fast Poisson solver para sistema híbrido reconfigurável**. 105 p. Dissertao (Mestrado) — Instituto Nacional de Pesquisas Espaciais (INPE), São José dos Campos, 2012-05-25 2012. Disponível em: <http://urlib.net/sid.inpe.br/mtc-m19/2012/05.10.19.52>. Acesso em: 28 jul. 2017. 38, 39, 42, 44

GOMES, V. C. F.; SAMBATTI, S. B.; FURTADO, H. C. M.; LUZ, E. F. P. d.; CHARÃO, A. S.; VELHO, H. F. d. C. Data assimilation by neural network on hardware device. In: INTERNATIONAL SYMPOSIUM ON INVERSE PROBLEMS, DESIGN AND OPTIMIZATION, 4. (IPDO)., 2013, Albi. **Proceedings...** Ecole des Mines d Albi, 2013. p. 01–08. Informações Adicionais: Abstract A6382HC. Disponível em: <http://ipdo2013.congres-scientifique.com/index.php?langue=en\&onglet=1\&acces=\&idUser=\&emailUser=>. Acesso em: 28 jul. 2017. 47, 50, 51

HADAMARD, J. **Lectures on Cauchy's problem in partial differential equations**. [S.l.]: Dover, 1952. 9

HÄRTER, F. P. **Redes neurais recorrentes aplicadas à assimilação de dados em dinâmica não-linear**. 138 p. Tese (Doutorado) — Instituto Nacional de Pesquisas Espaciais, São José dos Campos, 2004-12-16 2004. Disponível em: <http://urlib.net/sid.inpe.br/jeferson/2005/03.09.14.05>. Acesso em: 28 jul. 2017. 8, 10, 11, 81

HARTER, F. P.; Campos Velho, H. F. de. New approach to applying neural network in nonlinear dynamic model. **Applied Mathematical Modeling**, v. 32, n. 12, p. 2621 – 2633, 2008. 81

HAUCK, S.; DEHON, A. **Reconfigurable Computing:**: The theory and practice of fpga-based computation. [S.l.]: Morgan Kaufmann/Elsevier, 2008. 37, 39, 40, 43, 47

HAYKIN, S. **Redes Neurais:**: princípios e prática. [S.l.]: Prentice-Hall Inc., 2001. 15, 17, 18, 19, 20, 21, 22, 23, 24, 26

HEBB, D. O. **The organization of behavior:**: A neuropsychological theory. [S.l.]: Psychology Press, 1949. 16, 17

HINTON, G. E. Connectionist learning procedures. **Artificial Intelligence**, p. 185 – 234, 1989. 2

HOFFMAN, J. D.; FRANKEL, S. **Numerical methods for engineers and scientists**. [S.l.]: CRC press, 2001. 53

HOPFIELD, J. J. Neural networks and physical systems with emergent collective computational abilities. **Proceedings of the national academy of sciences**, National Acad Sciences, v. 79, n. 8, p. 2554–2558, 1982. 17

ISMAIL-ZADEH, A.; TACKLEY, P. **Computational methods for geodynamics**. [S.l.]: Cambridge University Press, 2010. 53, 54

KALNAY, E. **Atmospheric modeling, data assimilation, and predictability**. [S.l.]: Cambridge university press, 2003. 6, 9

LUCAS, K. The "all or none" contraction of the amphibian skeletal muscle fibre. **The Journal of physiology**, Wiley-Blackwell, v. 38, n. 2-3, p. 113, 1909. 16

LUZ, E. F. P. d. **Meta-heurísticas paralelas na solução de problemas inversos**. 155 p. Tese (Doutorado) — Instituto Nacional de Pesquisas Espaciais, São José dos Campos, 2011-03-08 2012. Disponível em: <http://urlib.net/sid.inpe.br/mtc-m19/2012/02.22.17.13>. Acesso em: 28 jul. 2017. 3, 29, 30, 32, 33, 34

LYNCH, L. **Numerical integration of linear and nonlinear wave equations**. Tese (Doutorado) — College of Florida Atlantic University, Florida, 2004. 54, 62

LYNCH, P. Weather prediction by numerical process. **The Emergence of Numerical Weather Prediction**, Cambridge University Press, p. 1–27, 2006. 5

MCCULLOCH, W. S.; PITTS, W. A logical calculus of the ideas immanent in nervous activity. **The bulletin of mathematical biophysics**, Springer, v. 5, n. 4, p. 115–133, 1943. 16

MENOTTI, R. **LALP:**: uma linguagem para exploração do paralelismo de loops em computação reconfigurável. Tese (Doutorado) — Instituto de Ciências Matemáticas e de Computação, 2010. 42

MESINGER, F.; ARAKAWA, A. Numerical methods used in atmospheric models, volume 1. In: **Global Atmospheric Research Program World**

Meteorological Organization, Geneva (Switzerland). [S.l.: s.n.], 1976. v. 1. 57, 58, 69

MOORE, G. Moore's law. **Electronics Magazine**, v. 38, n. 8, p. 114, 1965. 41

NOWOSAD, A. G. **Novas abordagens em assililação de dados meteorológicos**. 120 p. Tese (Doutorado) — Instituto Nacional de Pesquisas Espaciais (INPE), São José dos Campos, 2001-03-14 2001. Disponível em: <http://urlib.net/sid.inpe.br/jeferson/2004/03.11.13.50>. Acesso em: 28 jul. 2017. 8

OMONDI, A. R.; RAJAPAKSE, J. C. **FPGA implementations of neural networks**. New York, NY, USA: Springer New York, 2006. 49

RAISCH, D. **Computadores Híbridos, a próxima fronteira da computação**. 2010. Disponível em: <https://www.ibm.com/developerworks/community/blogs/tlcbr/entry/computadores_hibridos_a_proxima_fronteira_da_computacao?lang=en>. 37

RANDALL, D. A. The shallow water equations. **Department of Atmospheric Science, Colorado State University, Fort Collins**, 2006. 56

RICHARDSON, L. F. **Weather prediction by numerical process**. 2. ed. [S.l.]: Cambridge University Press, 2007. (Cambridge Mathematical Library). 5

ROSENBLATT, F. The perceptron: A probabilistic model for information storage and organization in the brain. **Psychological review**, American Psychological Association, v. 65, n. 6, p. 386, 1958. 17, 21

RUMELHART, D. E.; HINTON, G. E.; WILLIAMS, R. J. Learning representations by back-propagating errors. **Nature**, v. 323, n. 6088, p. 533–538, 1986. 17

SACCO, W. F.; KNUPP, D. C.; LUZ, E. F. P. d.; NETO, A. J. d. S. Algoritmo de colisão de partículas (particle collision algorithm). In: **Técnicas de Inteligência Computacional Inspiradas na Natureza - Aplicação em Problemas Inversos em Transferência Radiativa**. São Carlos, SP: SBMAC, 2009. v. 41, p. 79–89. ISBN 21753385. Acesso em: 28 jul. 2017. 30

SACCO, W. F.; OLIVEIRA, C. R. E. **A New Stochastic Optimization Algorithm based on a Particle Collision Metaheuristic**. 2005. 29, 30

SAMBATTI, S. B. M.; ANOCHI, J. A.; LUZ, E. F. P.; CARVALHO, A. R.; SHIGUEMORI, E. H.; Campos Velho, H. F. de. Automatic configuration for neural network applied to atmospheric temperature profile identification. In: 3RD INTERNATIONAL CONFERENCE ON INTERNATIONAL CONFERENCE ON ENGINEERING OPTIMIZATION, 3., 2012, Rio de Janeiro, Brazil. **Proceedings...** Rio de Janeiro, 2012. p. 1–9. 29

SAMBATTI, S. B. M.; ANOCHI, J. A.; LUZ, E. F. P.; CARVALHO, A. R.; SHIGUEMORI, E.; Campos Velho, H. F. de. Mpca meta-heuristics for automatic architecture optimization of a supervised artificial neural network. **Blucher Mechanical Engineering Proceedings**, v. 1, n. 1, p. 2930–2939, july 2012. 29

SAMPSON, J. J. **Some solutions of the shallow water wave equations**. [S.l.]: Swinburne University of Technology, Faculty of Engineering and Industrial Sciences, 2008. 56

SHIGUEMORI, E. H. **Recuperação de perfis de temperatura e umidade da atmosfera a partir de dados de satélite - abordagens por redes neurais artificiais e implementação em hardware**. Tese (Doutorado) — Instituto Nacional de Pesquisas Espaciais, São José dos Campos, 2007-05-22 2007. Disponível em: <http://urlib.net/sid.inpe.br/mtc-m17@80/2007/07.31.12.19>. Acesso em: 28 jul. 2017. 42

SKLIAROVA, I.; FERRARI, A. B. Introdução à computação reconfigurável. **Electrónica e Telecomunicações**, v. 4, n. 1, p. 103–119, 2003. 41

SORENSON, H. W. Least-squares estimation: from gauss to kalman. **IEEE spectrum**, IEEE, v. 7, n. 7, p. 63–68, 1970. 9

STONE, M. Cross-validation. a review. **Mathematicsche Operationforsch Statichen**, p. 127 – 139, 1978. 2

STRANG, G. **Introduction to applied mathematics**. [S.l.]: Wellesley-Cambridge Press, 1986. 9, 12

TALAGRAND, O.; COURTIER, P. Variational assimilation of meteorological observations with the adjoint vorticity equation. i: Theory. **Quarterly Journal of the Royal Meteorological Society**, Wiley Online Library, v. 113, n. 478, p. 1311–1328, 1987. 6, 9

TALBI, E.-G. **Metaheuristics:**: from design to implementation. [S.l.]: John Wiley & Sons, 2009. 28, 29

TODMAN, T. J.; CONSTANTINIDES, G. A.; WILTON, S. J.; MENCER, O.; LUK, W.; CHEUNG, P. Y. Reconfigurable computing: architectures and design methods. **IEE Proceedings-Computers and Digital Techniques**, IET, v. 152, n. 2, p. 193–207, 2005. 13, 39

TURING, A. M. Computing machinery and intelligence. **Mind**, JSTOR, v. 59, n. 236, p. 433–460, 1950. 15

ULMER, C.; HILLES, R.; THOMPSON, D. Reconfigurable computing aspects of the cray xd1. **Proceedings of the CUG 2005**, 2005. 44, 46, 47

VELHO, H. F. C. Problemas inversos:: Conceitos básicos e aplicações. **Boletim SBMAC**, Especial, n. 6, 2005. ISSN 1414-8374. Número Especial: IV Encontro de Modelagem Computacional, Nova Friburgo, RJ, nov/2001. Acesso em: 28 jul. 2017. 8

WAIN, R.; BUSH, I.; GUEST, M.; DEEGAN, M.; KOZIN, I.; KITCHEN, C. **An overview of FPGAs and FPGA programming:**: Initial experiences at daresbury. [S.l.]: Council for the Central Laboratory of the Research Councils, 2006. 40, 41

WEIGEND, A. S.; HUBERMAN, B. Predicting the future: a connectionist approach. **International Journal of Neural Systems**, p. 193 – 209, 1990. 2

WIDROW, B.; HOFF, M. E. et al. Adaptive switching circuits. In: NEW YORK. **IRE WESCON convention record**. [S.l.], 1960. v. 4, n. 1, p. 96–104. 21

WILLIAMS, D.; HINTON, G. Learning representations by back-propagating errors. **Nature**, v. 323, n. 6088, p. 533–538, 1986. 65

ANEXO

Código VHDL referente a um neurônio, representado pelo diagrama da Figura 4.7.

```
entity neuronio is
    generic (INPUT_SIZE : positive := 8);
    port(
        a           : in    std_logic_vector(15 downto 0);
        wr_e        : in    std_logic;
        peso_bias   : in    std_logic;
        w           : in    std_logic_vector(15 downto 0);
        vd          : in    std_logic;
        clr         : in    std_logic;
        rdy         : out   std_logic;
        v           : out   std_logic_vector(31 downto 0);
        clk         : in    std_logic;
        debug       : out   std_logic_vector(63 downto 0);
        reset_n     : in    std_logic
    );
end entity neuronio;


architecture basic of neuronio is

    signal s_weight : std_logic_vector(INPUT_SIZE*16-1 downto 0);
    signal s_v      : std_logic_vector(31 downto 0);
    signal s_bias   : std_logic_vector(15 downto 0);
    signal s_debug  : std_logic_vector(63 downto 0);

    component mult_acc is
        port(
            a       : in    std_logic_vector(15 downto 0);
            b       : in    std_logic_vector(15 downto 0);
            acc     : out   std_logic_vector(31 downto 0);
            enable  : in    std_logic;
            clr     : in    std_logic;
            bias    : in    std_logic_vector(15 downto 0);
            rdy     : out   std_logic;
            clk     : in    std_logic;
            reset_n : in    std_logic
        );
    end component;

begin
```

```
    weight_handler : process(clk, reset_n) is
    begin
        if (reset_n = '0') then
            s_weight    <=  (others => '0');
            s_debug     <= (others => '0');
        elsif (clk'event and clk = '1') then
            if (wr_e = '1') then
                if(peso_bias = '0') then
                    s_weight(INPUT_SIZE*16-1 downto 0) <= s_weight( (INPUT_SIZE-1)*16-1 downto 0)
                else
                    s_bias <= w;
                    s_debug(15 downto 0) <= w;
                    s_debug(63) <= '1';
                end if;
            elsif (vd = '1') then
                s_weight(INPUT_SIZE*16-1 downto 0) <= s_weight((INPUT_SIZE-1)*16-1 downto 0)
                & s_weight(INPUT_SIZE*16-1 downto (INPUT_SIZE-1)*16);
            end if;
        end if;
    end process weight_handler;

    mult_add_inst : mult_acc
        port map (
            a       => a,
            b       => s_weight(INPUT_SIZE*16-1 downto (INPUT_SIZE-1)*16),
            acc     => s_v,
            enable  => vd,
            clr     => clr,
            bias    => s_bias,
            rdy     => rdy,
            clk     => clk,
            reset_n => reset_n
        );


    v <= s_v;
    debug <= s_debug;
end architecture basic;
```

Código VHDL referente a camada de uma perceptron, representado pelo diagrama da Figura 4.8

```
entity mlp is
    generic (N_NEURONS_HIDDEN   : positive  := 8;
             INPUT_SIZE         : positive  := 8;
             N_NEURONS_OUTPUT   : positive  := 8);
    port(
        data        : in    std_logic_vector(15 downto 0);
        addr_peso   : in    std_logic_vector(N_NEURONS_HIDDEN+N_NEURONS_OUTPUT-1 downto 0);
        peso        : in    std_logic_vector(15 downto 0);
        peso_bias   : in    std_logic;
        vd          : in    std_logic;
        rdy         : out   std_logic;
        clr         : in    std_logic;
        u           : out   std_logic_vector(15 downto 0);
        clk         : in    std_logic;
        reset_n     : in    std_logic;
        debug       : out   std_logic_vector(63 downto 0);
        v_buffer    : out   std_logic_vector(255 downto 0);
        -- QDR 1
        ar_1        : out   std_logic_vector(19 downto 0);
        r_n_1       : out   std_logic;
        dr_1        : in    std_logic_vector(71 downto 0);
        -- QDR 2
        ar_2        : out   std_logic_vector(19 downto 0);
        r_n_2       : out   std_logic;
        dr_2        : in    std_logic_vector(71 downto 0)
    );
end entity mlp;

architecture basic of mlp is

component layer is
    generic (N_NEURONS    : positive  := 8;
             INPUT_SIZE   : positive  := 8);
    port(
        data        : in    std_logic_vector(15 downto 0);
        addr_peso   : in    std_logic_vector(N_NEURONS-1 downto 0);
        peso        : in    std_logic_vector(15 downto 0);
        peso_bias   : in    std_logic;
        vd          : in    std_logic;
        rdy         : out   std_logic;
        clr         : in    std_logic;
        u           : out   std_logic_vector(15 downto 0);
```

```
        clk         : in    std_logic;
        reset_n     : in    std_logic;
        debug       : out   std_logic_vector(63 downto 0);
        v_buffer    : out   std_logic_vector(255 downto 0);
        -- QDR
        ar          : out   std_logic_vector(19 downto 0);
        r_n         : out   std_logic;
        dr          : in    std_logic_vector(71 downto 0)
    );
end component;

    signal s_h_rdy      :   std_logic;
    signal s_h_u        :   std_logic_vector(15 downto 0);

    signal s_o_rdy      :   std_logic;
    signal s_o_u        :   std_logic_vector(15 downto 0);

    signal s_v_buffer   : std_logic_vector(255 downto 0);
    signal s_debug      : std_logic_vector(63 downto 0);
begin


    hidden_layer: layer
    generic map(N_NEURONS    => N_NEURONS_HIDDEN,
                INPUT_SIZE   => INPUT_SIZE)
    port map(
        data        => data,
        addr_peso   => addr_peso(N_NEURONS_HIDDEN-1 downto 0),
        peso        => peso,
        peso_bias   => peso_bias,
        vd          => vd,
        rdy         => s_h_rdy,
        clr         => clr,
        u           => s_h_u,
        clk         => clk,
        reset_n     => reset_n,
        debug       => open,
        v_buffer    => open,
        -- QDR
        ar          => ar_1,
        r_n         => r_n_1,
        dr          => dr_1
    );
```

```
    output_layer: layer
    generic map(N_NEURONS    => N_NEURONS_OUTPUT,
                INPUT_SIZE   => N_NEURONS_HIDDEN)
    port map(
        data        => s_h_u,
        addr_peso   => addr_peso(N_NEURONS_HIDDEN+N_NEURONS_OUTPUT-1 downto N_NEURONS_HIDDEN),
        peso        => peso,
        peso_bias   => peso_bias,
        vd          => s_h_rdy,
        rdy         => s_o_rdy,
        clr         => clr,
        u           => s_o_u,
        clk         => clk,
        reset_n     => reset_n,
        debug       => debug,
        v_buffer    => s_v_buffer,
        -- QDR
        ar          => ar_2,
        r_n         => r_n_2,
        dr          => dr_2
    );

    rdy <= s_o_rdy;
    u   <= s_o_u;
    v_buffer    <= s_v_buffer;

end architecture basic;
```

PUBLICAÇÕES TÉCNICO-CIENTÍFICAS EDITADAS PELO INPE

Teses e Dissertações (TDI)

Teses e Dissertações apresentadas nos Cursos de Pós-Graduação do INPE.

Manuais Técnicos (MAN)

São publicações de caráter técnico que incluem normas, procedimentos, instruções e orientações.

Notas Técnico-Científicas (NTC)

Incluem resultados preliminares de pesquisa, descrição de equipamentos, descrição e ou documentação de programas de computador, descrição de sistemas e experimentos, apresentação de testes, dados, atlas, e documentação de projetos de engenharia.

Relatórios de Pesquisa (RPQ)

Reportam resultados ou progressos de pesquisas tanto de natureza técnica quanto científica, cujo nível seja compatível com o de uma publicação em periódico nacional ou internacional.

Propostas e Relatórios de Projetos (PRP)

São propostas de projetos técnico-científicos e relatórios de acompanhamento de projetos, atividades e convênios.

Publicações Didáticas (PUD)

Incluem apostilas, notas de aula e manuais didáticos.

Publicações Seriadas

São os seriados técnico-científicos: boletins, periódicos, anuários e anais de eventos (simpósios e congressos). Constam destas publicações o Internacional Standard Serial Number (ISSN), que é um código único e definitivo para identificação de títulos de seriados.

Programas de Computador (PDC)

São a seqüência de instruções ou códigos, expressos em uma linguagem de programação compilada ou interpretada, a ser executada por um computador para alcançar um determinado objetivo. Aceitam-se tanto programas fonte quanto os executáveis.

Pré-publicações (PRE)

Todos os artigos publicados em periódicos, anais e como capítulos de livros.