Pós-Graduação em Ciência da Computação

NATÁLIA FLORA DE LIMA

FRANKENSTEIN PSO NA DEFINIÇÃO DAS ARQUITETURAS E AJUSTE DOS PESOS E USO DE PSO HETEROGÊNEO NO TREINAMENTO DE REDES NEURAIS FEED-FORWARD

Universidade Federal de Pernambuco
posgraduacao@cin.ufpe.br
www.cin.ufpe.br/~posgraduacao

RECIFE
2011

NATÁLIA FLORA DE LIMA

"FRANKENSTEIN PSO NA DEFINIÇÃO DAS ARQUITETURAS E AJUSTE DOS PESOS E USO DE PSO HETEROGÊNEO NO TREINAMENTO DE REDES NEURAIS FEED-FORWARD"

Dissertação apresentada à Pós-Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Mestre em Ciência da Computação.

ORIENTADORA: Profa. Dra. Teresa Bernarda Ludermir

RECIFE
2011

Catalogação na fonte
Bibliotecária Joana D'Arc Leão Salvador CRB 4-572

L732f Lima, Natália Flora de.
Frankenstein PSO na definição das arquiteturas e ajustes dos pesos e uso de PSO heterogêneo no treinamento de redes neurais feed-forward / Natália Flora de Lima. – 2011.
80 f.: fig., tab.

Orientadora: Teresa Bernarda Ludemir.
Dissertação (Mestrado) – Universidade Federal de Pernambuco. CIN. Ciência da Computação, Recife, 2011.
Inclui referências.
1. Inteligência artificial. 2. Redes neurais (Computação). I. Ludemir, Teresa Bernarda (Orientadora). II. Titulo.

006.3 CDD (22. ed.) UFPE-MEI 2016-72

Dissertação de Mestrado apresentada por **Natália Flora de Lima** à Pós-Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco, sob o título **"Frankenstein PSO na Definição das Arquiteturas e Ajuste dos Pesos e Uso de PSO Heterogêneo no Treinamento de Redes Neurais Feed-Forward"** orientada pela **Profa. Teresa Bernarda Ludermir** e aprovada pela Banca Examinadora formada pelos professores:

_____

Prof. Cleber Zanchettin
Centro de Informática / UFPE

_____

Prof. Wilson Rosa de Oliveira Junior
Departamento de Estatística e Informática /UFRPE

_____

Profa. Teresa Bernarda Ludermir
Centro de Informática / UFPE

Visto e permitida a impressão.
Recife, 29 de agosto de 2011.

_____

Prof. Nelson Souto Rosa
Coordenador da Pós-Graduação em Ciência da Computação do
Centro de Informática da Universidade Federal de Pernambuco.

Este trabalho é dedicado à minha mãe e amigos.

AGRADECIMENTOS

Agradeço à Prof.ª Dr.ª Teresa Bernarda Ludermir pela orientação, esclarecimentos e apoio no desenvolvimento deste trabalho e à amiga Teresa pelas palavras de conforto e apoio incondicional no período mais difícil de minha vida, fase em que perdi a minha mãe.

Obrigada a Fundação de Amparo à Ciência e Tecnologia do Estado de Pernambuco – FACEPE – pelo aporte financeiro, sem o qual não teria conseguido desenvolver este trabalho.

Não posso deixar de mencionar meus amigos e parceiros para todas as horas: Elaine Cristina, Danielle Nathália, Anderson Paulo, Adenilton, Cristiano, Breno, Maria Yêda e Renata Maria. O meu muito obrigada aos professores Cléber Zanchettin e principalmente a Wilson Rosa, que me acompanha desde a graduação, por participarem da minha banca. Ao professor Jones Albuquerque pelo incentivo à continuidade dos estudos. Agradecida a todos que direta ou indiretamente contribuíram com o desenvolvimento deste trabalho.

Por fim meu agradecimento especial a Helena Flora de Lima, por seu amor extremo, cuidado, por uma vida de dedicação e renúncias em prol da minha educação. Nunca poderia ter desejado uma mãe melhor, pena ela não estar mais ao meu lado, fisicamente, para acompanhar minhas vitórias. Mãe, eu te amo, obrigado Deus por ter me dado Helena de presente!

RESUMO

Este trabalho apresenta dois novos algoritmos, PSO-FPSO e FPSO-FPSO, para a otimização global de redes neurais MLP (do inglês *Multi Layer Perceptron*) do tipo *feed-forward*. O propósito destes algoritmos é otimizar de forma simultânea as arquiteturas e pesos sinápticos, objetivando melhorar a capacidade de generalização da rede neural artificial (RNA). O processo de otimização automática das arquiteturas e pesos de uma rede neural vem recebendo grande atenção na área de aprendizado supervisionado, principalmente em problemas de classificação de padrões. Além dos Algoritmos Genéticos, Busca Tabu, Evolução Diferencial, Recozimento simulado que comumente são empregados no treinamento de redes neurais podemos citar abordagens populacionais como a otimização por colônia de formigas, otimização por colônia de abelhas e otimização por enxame de partículas que vêm sendo largamente utilizadas nesta tarefa. A metodologia utilizada neste trabalho trata da aplicação de dois algoritmos do tipo PSO, sendo empregados na otimização das arquiteturas e na calibração dos pesos das conexões. Nesta abordagem os algoritmos são executados de forma alternada e por um número definido de vezes. Ainda no processo de ajuste dos pesos de uma rede neural MLP foram realizados experimentos com enxame de partículas heterogêneos, que nada mais é que a junção de dois ou mais PSOs de tipos diferentes. Para validar os experimentos com os enxames homogêneos foram utilizadas sete bases de dados para problemas de classificação de padrões, são elas: câncer, diabetes, coração, vidros, cavalos, soja e tireóide. Para os experimentos com enxames heterogêneos foram utilizadas três bases, a saber: câncer, diabetes e coração. O desempenho dos algoritmos foi medido pela média do erro percentual de classificação. Algoritmos da literatura são também considerados. Os resultados mostraram que os algoritmos investigados neste trabalho obtiveram melhor acurácia de classificação quando comparados com os algoritmos da literatura mencionados neste trabalho.

Palavras-chave: Otimização por Enxame de Partículas. Redes Neurais Artificiais. Enxames Heterogêneos.

ABSTRACT

This research presents two new algorithms, PSO-FPSO e FPSO-FPSO, that can be used in feed-forward MLP (Multi Layer Perceptron) neural networks for global optimization. The purpose of these algorithms is to optimize architectures and synaptic weight, at same time, to improve the capacity of generalization from Artificial Neural Network (ANN). The automatic optimization process of neural network's architectures and weights has received much attention in supervised learning, mainly in pattern classification problems. Besides the Genetic Algorithms, Tabu Search, Differential Evolution, Simulated Annealing that are commonly used in the training of neural networks we can mentioned population approaches such Ant Colony Optimization, Bee Colony Optimization and Particle Swarm Optimization that have been widely used this task. The methodology applied in this research reports the use of two PSO algorithms, used in architecture optimization and connection weight adjust. In this approach the algorithms are performed alternately and by predefined number of times. Still in the process of adjusting the weights of a MLP neural network experiments were performed with swarm of heterogeneous particles, which is nothing more than the joining of two or more different PSOs. To validate the experiments with homogeneous clusters were used seven databases for pattern classification problems, they are: cancer, diabetes, heart, glasses, horses, soy and thyroid. For the experiments with heterogeneous clusters were used three bases, namely cancer, diabetes and heart. The performance of the algorithms was measured by the average percentage of misclassification, literature algorithms are also considered. The results showed that the algorithms investigated in this research had better accuracy rating compared with some published algorithms.

Keywords: Particle Swarm Optimization. Artificial Neural Networks. Heterogeneous Swarm.

LISTA DE ALGORITMOS

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

Capítulo 1

Introdução

Este capítulo apresentará a motivação do trabalho, bem como os objetivos. Em seguida, a organização da dissertação é detalhada.

1.1 Motivação

Redes neurais artificiais (RNAs) têm obtido bons resultados na solução de vários problemas, como reconhecimento de padrões (CARVALHO; LUDERMIR, 2006c; YANAN; XIUWEI; LI, 2010), previsão de séries temporais (VALENÇA, 2010; VALENÇA; LUDERMIR; VALENÇA, 2010) aproximação de funções (GOMES; LUDERMIR, 2008; PRUDÊNCIO, 2002) e outros. Por este motivo existe um interesse crescente na aplicação de redes neurais em outras classes de problemas, bem como o refinamento de técnicas utilizadas em sua construção. Nem sempre o emprego de redes neurais produz bons resultados, muitas vezes isto se dá pela dificuldade em projetá-las a um problema específico. Por isto é cada vez mais frequente o uso das mais diversas técnicas na construção de redes neurais artificiais (ALMEIDA, 2011; LUDERMIR; YAMAZAKI; ZANCHETTIN, 2006).

Um dos modelos mais utilizados e conhecidos de redes neurais é o MLP (*Multi Layer Perceptron*) (BRAGA; CARVALHO; LUDERMIR, 2007; HAYKIN, 1999). Este tipo caracteriza-se por uma arquitetura disposta em camadas sendo a primeira de entrada – definida baseada na dimensão do problema a ser tratado – uma ou mais camadas ocultas, também chamadas de camadas intermediárias e uma camada de saída – definida de acordo com o problema a ser resolvido. Definir a configuração da(s) camada(s) intermediária(s) é uma tarefa complexa, pois como eleger a quantidade ideal de camadas e o número de neurônios que cada uma delas irá possuir, senão pela tentativa e erro ou consulta a um especialista. Esta é uma das razões pela qual o uso de técnicas inteligentes combinadas a redes neurais tem feito tanto sucesso, comumente estas técnicas também têm sido empregadas na escolha de funções de

ativação, algoritmo de treinamento e outros parâmetros das redes neurais segundo Almeida e Ludermir (2009).

O processo de treinamento de redes neurais artificiais (RNAs) MLP para problemas de classificação de padrões envolvem duas fases. A primeira é a definição do número de camadas ocultas e a quantidade de neurônios que cada uma irá possuir, enquanto que a segunda fase trata de ajustar os pesos das conexões, conforme Braga, Carvalho e Ludermir (2007). Para definir uma arquitetura o menos complexa possível, podem ser empregadas técnicas inteligentes como Evolução Diferencial (SILVA; MINEU; LUDERMIR, 2009), Colônia de Formigas (SIVAGAMINATHAN; RAMAKRISHNAN, 2007), Algoritmos Genéticos (ALMEIDA; LUDERMIR, 2009; ZANCHETTIN; LUDERMIR, 2005), Otimização por Enxame de Partículas (CARVALHO; LUDERMIR, 2007; LUDERMIR; KIRANYAZ et al, 2009; YAMAZAKI; ZANCHETTIN, 2006), Colônia de Abelhas (KARABOGA; BASTURK, 2008; TEODOROVIC et al, 2006), Programação Evolucionária (YAO; LIU, 1997). Estas técnicas também podem ser utilizadas no ajuste dos pesos das conexões da rede para substituir o conhecido algoritmo *back-propagation*, pois o mesmo utiliza o gradiente descendente do erro de classificação para determinar o novo valor dos pesos das conexões. O *back-propagation* faz uso do *aprendizado online,* o que quer dizer que, geralmente, todos os padrões de treino precisam ser apresentados de forma continua à medida que a fase de treinamento avança. Segundo Braga, Carvalho e Ludermir (2007) o uso do *back-propagation* em redes neurais grandes e/ou complexas torna-se muito difícil. Outro ponto negativo em sua utilização, que ocorre durante a fase de treinamento, é o *overfitting*. Isto ocorre quando a rede passa a memorizar os padrões de entrada apresentados, diminuindo assim a capacidade de generalização da rede neural. Para resolver este problema reserva-se uma parte do conjunto de treinamento, geralmente 25% dos dados. Este subconjunto é chamado *conjunto de validação* e é utilizado para avaliar se durante a fase de treinamento está ocorrendo o *overfitting*. O treinamento da rede neural é interrompido a partir do momento que o erro no conjunto de validação começa a crescer. Por outro lado, se o treinamento for interrompido muito cedo poderá ocorrer o chamado *underfitting,* que é a incapacidade de generalização da rede, ou seja, falta de conexões e/ou parâmetros ajustáveis conforme Braga, Carvalho e Ludermir (2007).

Alguns trabalhos tem considerado o uso Algoritmos Genéticos (AG) como Almeida e Ludermir (2009); Zanchettin e Ludermir (2009); Evolução Diferencial (ED) como Zarth (2010); Zarth e Ludermir (2009); Busca Tabu (BT) conforme Zanchettin, Ludermir e Almeida (2011); Recozimento Simulado (RS) segundo Yamazaki (2004); Zanchettin, Ludermir e Almeida (2011) ou mesmo duas ou mais destas técnicas juntas para definir a configuração de redes neurais, como é o caso de Almeida e Ludermir (2009) que combinou Estratégias de Evolução (*Evolution Strategy*), Algoritmos Genéticos e Otimização por Enxame de Partículas para aperfeiçoar uma RNA através da escolha do algoritmo de treinamento, do pesos das conexões, da taxa de aprendizado, do número de camadas intermediárias, das funções de transferência e do bias. No trabalho Zarth (2010) o sistema proposto buscou por arquiteturas e pesos fazendo uso de Evolução Diferencial associado a uma estratégia para controle da diversidade proveniente da Computação Evolucionária Paralela. Em Eberhart e Shi (2000) foram combinadas as técnicas de Recozimento Simulado, Busca Tabu e o algoritmo por correção de erro *back-propagation* com o objetivo de gerar redes neurais de baixa complexidade e alta capacidade de generalização.

O uso de Otimização por Enxame de Partículas (PSO) tem se tornado cada vez mais frequente em estudos relacionados ao aprendizado supervisionado. Esta técnica tenta simular o comportamento animal na busca por recursos, por exemplo, imagine um bando de pássaros a procura de novas fontes de alimentação ou um local de descanso. Neste cenário os pássaros são representados pelas partículas, as fontes de alimentação ou o local de descanso é a função objetivo e a área onde os pássaros se deslocam representa o espaço de busca. O PSO foi apresentado pela primeira vez por Kennedy e Eberhart (1995) e desde então tem sido utilizado na melhoria de soluções de vários problemas na área de redes neurais artificiais (CHAURASIA; DAWARE, 2009; GUDISE; VENAYAGAMOORTHY, 2003; ZHONG; WANG; LI, 2009), tendo alcançado bons resultados quando empregado a problemas de otimização numérica. Além disso o PSO é considerado um método de fácil implementação.

Este trabalho faz uso da Otimização por Enxame de Partículas para definir o número de neurônios na camada intermediária e realizar o treinamento de uma rede MLP, do tipo *feed-forward*.

1.2 Objetivos

Além de fazer uso do PSO para definir redes neurais de baixa complexidade e boa capacidade de generalização, os seguintes tópicos podem ser destacados como objetivos deste trabalho:

- Avaliar o desempenho de uma variação recente da técnica de otimização por enxame de partículas, chamado Frankenstein PSO ou apenas FPSO, em relação à técnica padrão quando aplicado a problemas de classificação de padrões.
- Validar novos algoritmos de ajuste simultâneo das arquiteturas e pesos de uma rede neural, cuja metodologia é similar a apresentada em Carvalho (2007).
- Legitimar o uso de enxames heterogêneos no processo de treinamento de redes neurais *Multi Layer Perceptron*.

1.3 Organização da Dissertação

Esta dissertação está organizada em 6 capítulos, estando no capítulo primeiro a introdução, motivação, objetivos e organização deste trabalho. Os demais capítulos seguem organizados da seguinte forma:

- Capítulo 2 – Otimização de Redes Neurais Utilizando Enxame de Partículas: neste capítulo apresentamos o conceito de neurônio e rede neural artificial e alguns métodos baseados na otimização por enxame de partículas, utilizados neste trabalho para otimizar o treinamento e a definição da arquitetura de redes neurais do tipo *feed-forward*.
- Capítulo 3 – Treinamento de Redes Neurais com PSO: É descrito o uso do Frankenstein PSO no processo de otimização dos pesos das conexões em problemas de classificação de padrões.
- Capítulo 4 – Ajuste Simultâneo de Pesos e Arquiteturas com FPSO: Trata do ajuste simultâneo dos pesos e arquiteturas fazendo uso do Frankenstein PSO. Foram utilizados os resultados obtidos no capítulo 3 e as mesmas bases de dados para realizar os experimentos.

- Capítulo 5 – Enxames Heterogêneos: Apresenta o uso de enxames heterogêneos no treinamento de redes neurais MLP.
- Capítulo 6 – Conclusões e Trabalhos Futuros: Aborda as conclusões e também cita alguns trabalhos futuros.

Capítulo 2

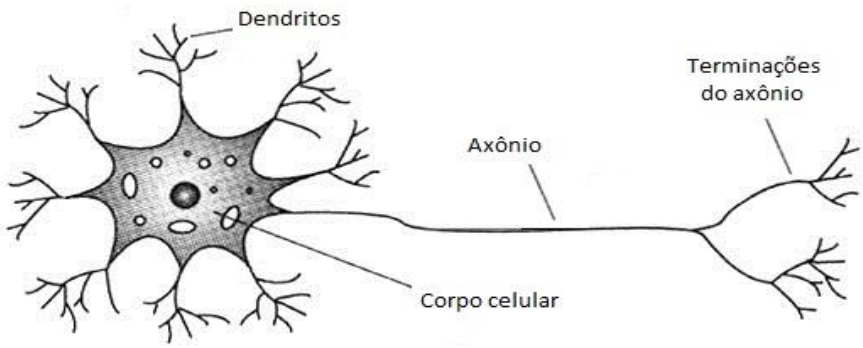
Otimização de Redes Neurais Utilizando Enxame de Partículas

Neste capítulo é apresentada a definição de neurônio e redes neurais artificiais, PSO padrão e algumas variações da otimização por enxame de partículas utilizadas neste trabalho.

2.1 Redes Neurais Artificiais

Redes Neurais Artificiais são representações computacionais, em contínuo aperfeiçoamento, que tentam reproduzir o processo de aprendizagem do cérebro humano. Assim como o cérebro que é formado por neurônios as redes neurais também o possuem em sua constituição. Os neurônios ou nodos, como também são conhecidos, são associados aos demais por meio de conexões, que por sua vez recebem um peso. O comportamento ou resposta de uma rede neural a determinado estímulo é dado pela força destas conexões (BRAGA; CARVALHO; LUDERMIR, 2007). Isto significa que, quanto maior o peso atribuído a uma determinada conexão, maior será sua influência na saída (resposta). Na Figura 1 temos uma representação gráfica de um neurônio biológico.

Figura 1: Neurônio biológico.



Fonte: Autor (2011).

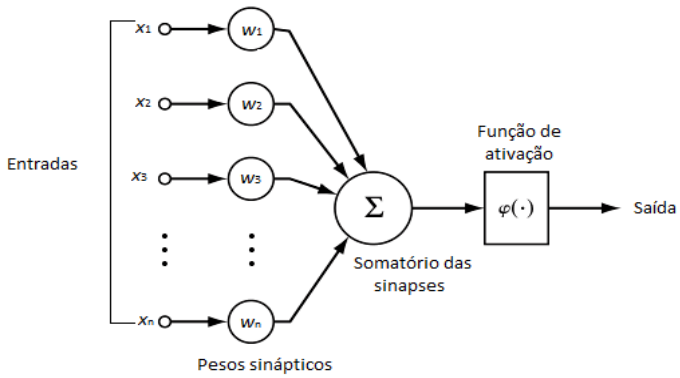
De modo genérico podemos descrever a formação e funcionamento de um neurônio biológico da seguinte forma: os estímulos de entrada, provenientes de outros neurônios, são captados por meio dos *dendritos*; o estímulo, então, é processado no *corpo celular* e a saída ou resposta será transmitida ao próximo neurônio por meio do *axônio*.

Na década de 40 uma simplificada representação do comportamento de um neurônio biológico foi proposta por McCulloch e Pitts (BRAGA; CARVALHO; LUDERMIR, 2007; HAYKIN, 1999). No modelo de neurônio artificial MCP ou Modelo McCulloch-Pitts, os dendritos são representados por *n* terminais de entrada, $x_1, x_2, ..., x_n$; o axônio (que representa a saída ou resposta) simbolizado por uma única saída, *y*. Para representar o processo de transmissão da resposta de um neurônio a outro – sinapse – as conexões de entrada receberam pesos, $w_1, w_2, ..., w_3$. Deste modo para saber se o resultado aos estímulos de entrada recebidos foi suficiente para atingir o limiar de excitação do neurônio, *threshold*, é preciso somar todas as entradas aos respectivos pesos, que podem assumir valores positivos ou negativos. Logo, obtemos a equação (1), representação matemática do Modelo McCulloch-Pitts.

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

em que θ representa o limiar de excitação ou *threshold*. Podemos representar o neurônio artificial, proveniente do Modelo McCulloch-Pitts, conforme a Figura 2.

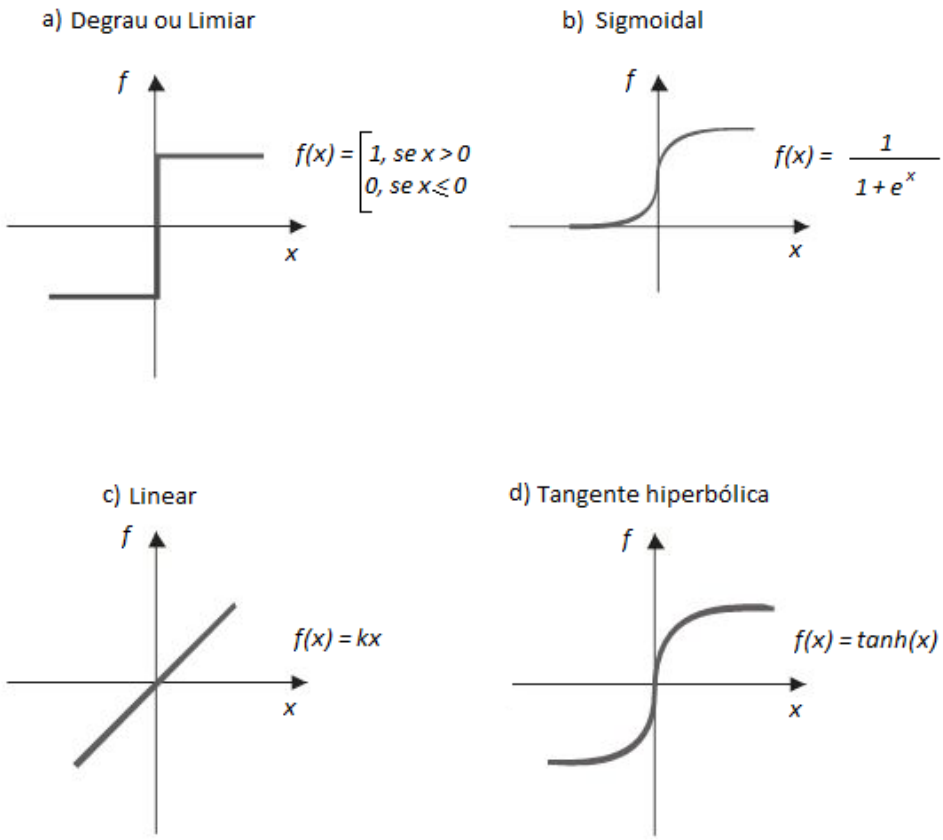
Figura 2: Esquematização de um neurônio artificial MCP.



Fonte: Braga, Carvalho e Ludermir (2007).

No modelo MCP os neurônios são binários, ou seja, possuem saída 0 ou 1, então, para determinado estímulo de entrada o neurônio estará ativo ou não. No entanto a grande maioria dos problemas da vida real são não-lineares. Com o intuito de solucionar esta limitação, ao longo dos anos, foram apresentadas diferentes funções de ativação, algumas delas estão dispostas na Figura 3.

Figura 3: Exemplos de funções de ativação.



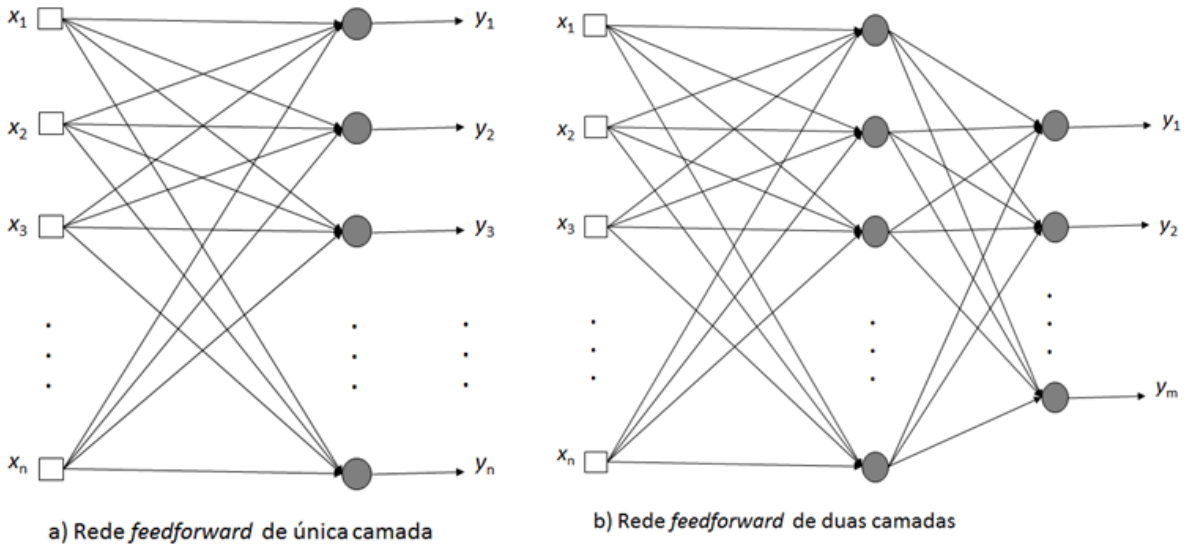
Fonte: Braga, Carvalho e Ludermir (2007).

Um único neurônio possui capacidade computacional bastante limitada, no entanto se juntarmos vários neurônios e os dispormos em forma de rede teremos a capacidade de resolver problemas de alta complexidade. Na Redes neurais de única camada, como visualizado em Figura 4. a) são capazes de resolver apenas problemas lineares. Na arquitetura disposta em Figura 4. b) também temos uma rede *feed-forward*, no entanto esta possui uma camada adicional, a chamada camada intermediária ou camada oculta. Redes neurais com esta disposição de neurônios são capazes de resolver qualquer função contínua. Se adicionarmos mais uma camada intermediária

a esta teremos uma rede capaz de aproximar qualquer função conforme Braga, Carvalho e Ludermir (2007) e mais, se os neurônios da camada intermediária fizerem uso de funções sigmoidais esta rede receberá o nome de Perceptron de Múltiplas Camadas ou MLP, do inglês Multi Layer Perceptron. Na Figura 4 são apresentadas apenas as arquiteturas utilizadas neste trabalho, redes *feed-forward*.

Redes neurais de única camada, como visualizado em Figura 4 a) são capazes de resolver apenas problemas lineares. Na arquitetura disposta em Figura 4 b) também temos uma rede *feed-forward*, no entanto esta possui uma camada adicional, a chamada camada intermediária ou camada oculta. Redes neurais com esta disposição de neurônios são capazes de resolver qualquer função contínua. Segundo Braga, Carvalho e Ludermir (2007) se adicionarmos mais uma camada intermediária a esta teremos uma rede capaz de aproximar qualquer função e mais, se os neurônios da camada intermediária fizerem uso de funções sigmoidais esta rede receberá o nome de Perceptron de Múltiplas Camadas ou MLP, do inglês Multi Layer Perceptron.

Figura 4: Arquitetura de uma rede neural *feed-forward*.



Fonte: Braga, Carvalho e Ludermir (2007).

Uma vez definidos a arquitetura e função ou funções de ativação da rede neural é preciso treiná-la para que possa “aprender”. Dentre as diversas formas de aprendizado aplicáveis as redes neurais (por competição, reforço, hebbiano e outros) veremos de forma detalhada o aprendizado supervisionado, que é o método utilizado neste trabalho.

Neste tipo de aprendizado uma parte do conjunto de dados é reservada para realizar o treinamento da rede. A rede deve calcular o quão distante da resposta desejada ela se encontra, ou seja, para cada saída obtida e de acordo com a resposta desejada a rede calcula o erro da saída e assim, ajusta os pesos de suas conexões, de forma que a resposta da rede se aproxime da saída desejada.

Apesar dos bons resultados obtidos pelas redes neurais em tarefas de classificação, categorização, otimização, aproximação e previsão existem alguns pontos que contam negativamente em sua aplicação. A exemplo podemos citar a definição da quantidade ideal dos dados de entrada utilizados pelo conjunto de treinamento; a duração da fase de treinamento ou aprendizagem, podendo causar problemas como *overfitting* (quando a rede passa a “decorar” os dados de entrada produzindo altas taxas de erro na fase de teste) e *underfitting* (quando a fase de treinamento é muito curta e incapacita a rede a regular os parâmetros ajustáveis. Também produz alta taxa de erro durante a fase de teste); o uso de algoritmos por correção de erro, como o *back-propagation*, que podem levar a rede a regiões de mínimos locais. Algumas soluções a estes e outros problemas é a combinação de diferentes métodos de forma que um seja capaz de suprir as deficiências do outro ou mesmo para encontrar soluções mais eficientes e/ou robustas. A esta combinação de diferentes métodos, no qual um deles seja uma rede neural, damos o nome de Sistemas Neurais Híbridos (BRAGA; CARVALHO; LUDERMIR, 2007).

2.2 PSO Padrão

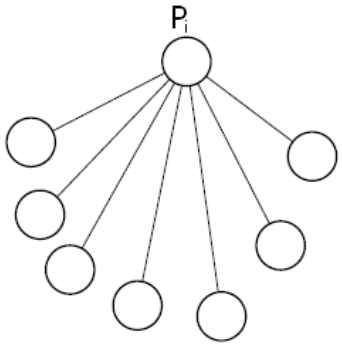
A Otimização por Enxame de Partículas ou apenas PSO é uma técnica meta-heurística criada por Kennedy e Eberhart (1995) cuja motivação partiu da observação do comportamento social de animais. Técnicas baseadas no comportamento de populações são largamente aplicadas a um grande número de problemas de otimização numérica. A exemplo podemos citar a Otimização por Colônia de Formigas (do inglês *Ant Colony Optimization*) em Sivagaminathan e Ramakrishnan (2007) e a Otimização por Colônia de Abelhas (do inglês *Bee Colony Optimization*) em Karaboga e Basturk, (2008); Teodorovic et al, (2006). A idéia inicial dos autores foi reproduzir o comportamento de pássaros na busca por recursos, por exemplo, a busca por novas

fontes de alimentação. Assim como na natureza em que os indivíduos se movimentam no espaço trocando informações entre si com o objetivo de levar o bando as melhores regiões o algoritmo do PSO também reproduz este comportamento. A seguir descrevemos como este hábito foi modelado na otimização por enxame de partículas.

Seja o enxame um conjunto *S* formado por partículas *p*, que representam as possíveis soluções, cada uma delas tem dimensão *n*. Para cada uma delas, $1 \leq p_i \leq S$, num instante de tempo *t*, tem suas posições $x_i(t) \in \Re n$ e velocidades $v_i(t) \in \Re n$ definidas - determinam a extensão e direção do deslocamento no espaço de soluções. Informações adicionais são armazenadas pelas variáveis *pb* – responsável por guardar a melhor posição visitada pela partícula – e *gb* – armazena a melhor posição visitada pelo enxame.

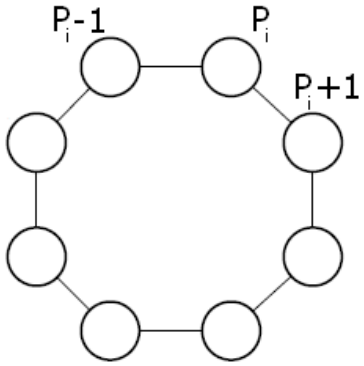
Outra particularidade do PSO é a maneira como às partículas trocam informações entre si. Originalmente a otimização por enxame de partículas utiliza um modelo conhecido por *Gbest – global best*. Neste modelo de vizinhança a partícula do enxame que alcançou a melhor posição no espaço de busca, ou seja, a melhor solução até o momento, influência o deslocamento das demais partículas. A adoção do *Gbest* proporciona uma convergência mais rápida, uma vez que a propagação da informação é ágil. Por outro lado, existe um ponto negativo, a possibilidade de convergência prematura, que pode fazer com que o enxame se desloque a uma região sub-ótima, mínimo local. Visando minimizar o problema da convergência prematura, foi criado o modelo *Lbest – local best*, também conhecido por *Anel*. Neste modelo topológico uma partícula influência apenas algumas de suas vizinhas (normalmente adota-se vizinhança de grau 1, o que significa que a partícula que alcançou a melhor posição no espaço de buscas influência suas vizinhas imediatas. No entanto, pode-se adotar um grau de vizinhança diferente). A Figura 5 e Figura 6 exibem uma representação dos modelos de vizinhança *Gbest* e *Lbest*, respectivamente.

Figura 5: Modelo de vizinhança Gbest



Fonte: Autor (2011).

Figura 6: Modelo de vizinhança Lbest.



Fonte: Autor (2011).

Para calcular a velocidade das partículas, utiliza-se a equação (2) e para atualizar a posição a equação (3). Nas duas equações o índice *i* representa a componente, no instante *t*, do vetor *n*.

$$v_i(t+1) = v_i(t) + c_1 r_1(pb_i(t) - x_i(t)) + c_2 r_2(gb_i(t) - x_i(t)) \quad (2)$$

$$x_i(t+1) = x_i(t) + v_i(t+1) \quad (3)$$

Algumas variações do PSO fazem uso do peso de inércia (*w*) (EBERHART; SHI, 2001; MONTES DE OCA et al, 2009a; SHI; EBERHART, 1999; ZHENG et al, 2003), proposto por Shi e Eberhart (1998). Este fator multiplica a velocidade da partícula, no instante *t*, proporcionando aumento da exploração (tenta cobrir a maior área possível do espaço de busca) nas primeiras iterações e explotação (cobre pequenas áreas das regiões mais promissoras do espaço de busca) nas iterações finais, de acordo com o valor que a variável *w* assume ao longo das iterações. Em Shi e Eberhart (1999) o valor do peso de inércia varia de forma linear e decrescente dentro do intervalo de 0.9 a 0.4. Deste modo a equação (2), utilizada para controlar o movimento das partículas no espaço de busca, transforma-se na equação (4):

$$v_i(t+1) = wv_i(t) + c_1 r_1(pb_i(t) - x_i(t)) + c_2 r_2(gb_i(t) - x_i(t)) \quad (4)$$

Tanto na equação (2) quanto na equação (4) as variáveis r_1 e r_2 assumem valores randômicos gerados no intervalo [0,1], estes valores são gerados a cada iteração para determinar a influência dos fatores individuais e global na iteração (*pb* e *gb*,

respectivamente). As variáveis c_1 e c_2 representam os coeficientes de aceleração que, por sua vez, possuem valores fixos e iguais, $c_1 = c_2 = 2{,}05$, valores disponíveis em Clerc e Kennedy (2002).

Algoritmo 1: PSO padrão.

```
1:  Iniciar randomicamente a população de partículas, P
2:  Repita
3:      Para cada partícula p_i da população P faça
4:          Se f(x_i(t)) < f(pb_i(t)) então
5:              pb_i(t) = x_i(t)
6:          Fim do se
7:          Se f(pb_i(t)) < f(gb_i(t))
8:              gb_i(t) = pb_i(t)
9:          Fim do se
10:     Fim do para
11:     Atualizar a velocidade e posição de p_i conforme as equações (3) e (2)
        respectivamente.
12: Até critério de parada ser satisfeito
```

Fonte: Montes de Oca et al (2009a).

No Algoritmo 1 apresentamos o pseudo-código utilizado para representar o algoritmo do PSO padrão. O funcionamento geral do algoritmo pode ser entendido da seguinte forma: inicialmente as partículas tem suas posições e velocidades determinadas de forma randômica (passo 1). A partir daí segue-se o processo de avaliação das partículas. Inicialmente verifica-se a melhor posição alcançada e seu valor é armazenado na variável pb (passo 4 e 5). A seguir a mesma verificação é feita, sendo agora em nível de enxame, gb (passo 7 e 8). O próximo passo é calcular as posições e velocidades para a iteração seguinte (passo 11). A avaliação das partículas repete-se até que um critério de parada seja alcançado, definido pela função objetivo - que pode ser de maximização ou minimização - ou até que o número máximo de iterações seja alcançado (passo 12).

2.3 Frankenstein PSO

Este novo algoritmo de otimização por enxame de partículas, chamado Frankenstein PSO ou apenas FPSO foi proposto por Montes de Oca et al (2009a) e surgiu a partir do interesse dos autores em avaliar a combinação de diversas variações do algoritmo PSO apresentados a partir da proposta inicial por Kennedy e Eberhart (1995). Para compor o FPSO foram analisadas sete variantes da técnica de otimização por enxame de partículas, são elas:

- *Constricted Particle Swarm Optimizer* – Trata-se de um fator contração adicionado à regra de atualização da velocidade por Clerc e Kennedy (2002) para evitar o crescimento descontrolado da velocidade da partícula, assim elas não ultrapassam o limite da área de busca. Deste modo a regra de atualização da velocidade foi modificada para equação (5):

$$v_i(t+1) = X(v_i(t) + c_1 U_i(t)(pb_i(t)\ tz - x_i(t)) + c_2\ U_i(t)(gb_i(t) - x_i(t))) \quad (5)$$

$$X = 2 / |2 - c - \sqrt[2]{c2 - 4c}| \quad (6)$$

onde *X* é o fator de contração definido pela equação (6) e a constante *c* possui valor igual a 2.05.

- *Time-Decreasing Inertia Weight Variant*, Shi e Eberhart (1999) propuseram definir o peso de inércia de acordo com uma função que faz o valor variar de forma decrescente. Deste modo nas primeiras iterações o algoritmo explora o espaço de busca e só depois foca nas regiões mais promissoras. A função usada para calcular o valor do peso de inércia é a equação (7):

$$W_t = ((w_{tmax} - t) / w_{tmax})*(w_{max} - w_{min}) + w_{min} \quad (7)$$

  onde W_{tmax} marca o momento em que $W_t = W_{min}$, normalmente W_{tmax} coincide com o tempo máximo alocado para o processo de otimização.

- *Increasing Inertia Weight Particle Swarm Optimization* Zheng et al (2003), esta variação da otimização por enxame de partículas é o inverso da proposta anterior, *Time-Decreasing Inertia Weight Variant*. Aqui a mesma fórmula de atualização do

peso de inércia foi adotada, exceto pelo fato de os valores das variáveis w_{max} e w_{min} serem invertidos.

- *Stochastic Inertia Weight Particle Swarm Optimization,* Eberhart e Shi (2001), nesta variante o vetor do peso de inércia é gerado randomicamente de acordo com uma distribuição uniforme definida no intervalo de [0.5, 1.0) com o peso de inércia diferente para cada dimensão. Neste algoritmo os coeficientes de aceleração são definidos pelo produto de $x * \varphi_i$, sendo i € {1, 2}.

- *Fully Informed Particle Swarm Optimizer* – FIPS, criado por Mendes, Kennedy e Neves (2004), responsável por considerar o número de vizinhas topológicas no processo de atualização da velocidade da partícula. Esta abordagem produz, então, uma nova equação para o cálculo da velocidade da partícula, a equação (8):

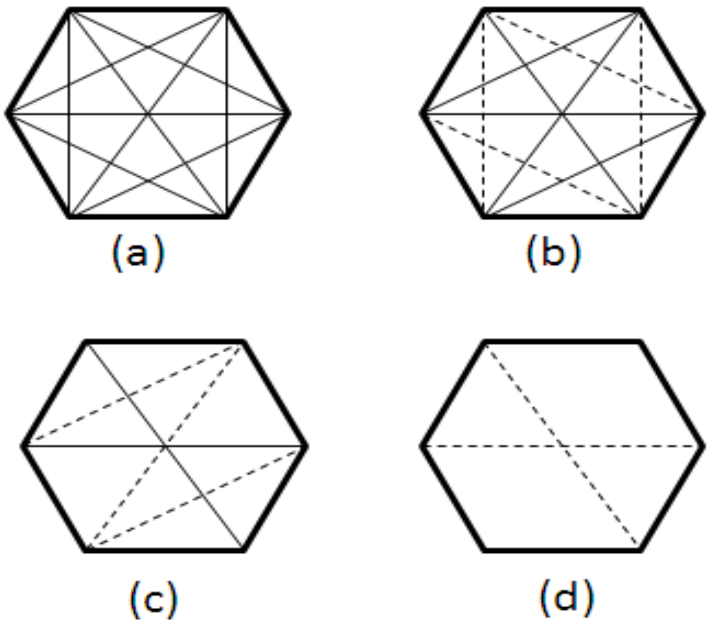
$$v_i(t+1) = wv_i(t) + \Sigma_{Pm \in Ni}\, \varphi_k U_k(t)\, (pb_k(t) - x_i(t)) \qquad (8)$$

  no FIPS o fator de contração de Clerc e Kennedy normalmente tem seu valor igual a 4.1 Eberhart e Shi (2000), o φ (soma dos coeficientes de aceleração) é distribuído igualmente dentre as partículas vizinhas.

- *Self-Organizing Hierarchical PSO With Time-varying Acceleration Coefficients*, nesta variação da otimização por enxame de partículas o termo de inércia na regra de atualização da velocidade é eliminado. No entanto quando algum componente do vetor da velocidade da partícula assume valor zero ou muito próximo a zero seu valor é reinicializado proporcionalmente ao definido em V_{max}. Esta reinicialização dá ao algoritmo um comportamento de busca local no qual os valores dos coeficientes de aceleração são adaptados linearmente, ou seja, o valor do coeficiente φ_1 é decrementado de 2.5 até 0.5 e o coeficiente φ_2, o inverso, o valor é incrementado de 0.5 a 2.5. Próximo ao final da execução o algoritmo atinge uma velocidade baixa, isto possibilita que as partículas se movam lentamente no entorno da melhor região encontrada. Esta proposta de PSO foi apresentada em (Ratnaweera, Halgamuge e Watson (2004).

- *Adaptive Hierarchical Particle Swarm Optimizer* ou AHPSO por Middendorf e Janson (2005), responsável pelo grau de conectividade entre as partículas, ou seja, a topologia da rede - o modo como as partículas propagam as informações entre si. Este processo pode ser visualizado na Figura 7.

Figura 7: Processo de atualização da topologia proposta pelo algoritmo AHPSO.



Fonte: Montes de Oca et al (2009a).

O processo de atualização da topologia dá-se da seguinte forma: inicialmente as partículas do FPSO ou aquelas que seguem o modelo AHPSO são completamente conectadas, ou seja, todas as partículas são vizinhas entre si - neste trabalho uma partícula é vizinha de si mesma. Ao longo das iterações algumas conexões são removidas dando lugar, ao final do processo de atualização, a topologia anel. Suponha que temos n partículas, as n(n-1)/2 arestas da topologia completamente conectada inicialmente darão lugar a apenas n, da topologia anel, em n-3 passos. Este processo de remoção é realizado seguindo um padrão de regressão aritmética, assim a cada iteração um número decrescente de arestas é removido. Na Figura 7 n = 6, então em 3 passos o processo de transformação da topologia completamente conectada em anel estará concluído. Vamos supor k = 12 (parâmetro utilizado no cálculo para indicar quando as arestas serão removidas, seu uso pode ser visualizado no algoritmo do FPSO, Algoritmo 2), no passo (a) o grafo é completamente conectado; no passo (b) 4

arestas são removidas (linhas pontilhadas); no passo (c) outras 3 arestas são removidas e por fim o passo (d) as 2 últimas arestas são removidas dando fim ao processo de atualização da topologia.

O resultado final da análise destas 7 variantes de PSO foi um algoritmo composto formado a partir de três componentes algorítmicos. Os componentes utilizados nesta composição foram: *Adaptive Hierarchical Particle Swarm Optimizer*, *Fully Informed Particle Swarm Optimizer* e *Time-Decreasing Inertia Weight Variant*. O funcionamento do algoritmo do Frankenstein PSO é análogo ao do PSO padrão. Podendo ser sistematizado da seguinte forma:

As partículas têm suas posições e velocidades, inicialmente, criadas de forma randômica; as melhores posições já visitadas pelas partículas, pb_i, recebem as recém-criadas posições randômicas, logo após o vetor de vizinhança das partículas é preenchido.

As variáveis de controle são inicializadas e então entra-se no laço principal, dentro dele verifica-se, para cada partícula, a melhor posição visitada atualizando pb_i quando necessário.

A seguir temos os três componentes algoritmos oriundos de outras versões de PSO, o módulo de controle da topologia (Algoritmo 2.2, passo 14), seguido do módulo que calcula o valor do peso de inércia para a iteração (Algoritmo 2.2, passo 23) e por último o que incorpora o número de vizinhas topológicas no cálculo da velocidade (Algoritmo 2.2, passo 28).

Por fim, (Algoritmo 2.2, passos 35 e 36), assim como no algoritmo do PSO padrão são verificados os critérios de parada, função objetivo e número de iterações.

As instruções do Frankenstein PSO estão dispostas no Algoritmo 2.

Algoritmo 2: Frankenstein PSO.

1: Para i = 1 a n faça
2: Criar a partícula p_i e adicioná-la ao conjunto de partículas P
3: Inicialize seus vetores x_i e v_i com valores randômicos dentro do espaço de busca e velocidade máximas permitidas
Definir $pb_i = x_i$
4: Definir $N_i = P$
5: Fim do para
6: Definir $t = 0$
7: Definir *steps* = 0
8: Repetir
9: Para i = 1 a n faça
10: Se $f(x_i)$ é melhor que $f(pb_i)$
11: Definir $pb_i = x_i$
12: Fim do se
13: Fim do para
14: Se$t > 0 \wedge t <= k \wedge \text{mod}\lceil k/(n-3)\rceil = 0$ então
15: Para i = 1 to n – (2 + *steps*) faça
16: Se $| N_i | > 2$ então
17: Elimine a partícula p_r de N_i
18: Elimine a partícula p_i de N_i
19: Fim do se
20: Fim do para
21: Definir *steps* = *steps* + 1
22: Fim do se
23: Se $t \leq wt_{max}$ então
24: Definir $w^t = ((w_{tmax} – t) / w_{tmax})*(w_{max} - w_{min}) + w_{min}$
25: Senão
26: Definir $w^t = w_{min}$
27: Fim do se
28: Para i = 1 to n faça
29: Gerar $U_m(t) \; \forall p_m \in N_i$
30: Definir $\varphi_m = \varphi / | N_i | \; \forall p_m \in N_i$
31: Definir $v_i(t+1) = w(t)v_i(t) + \Sigma_{Pm \in Ni} \varphi_k U_k(t) (pb_k(t) – x_i(t))$
32: Definir $x_i(t+1) = x_i(t) + v_i(t+1)$
33: Fim do para
34: Definir $t = t + 1$
35: Definir *solução* $=\text{argmin}_{pi \in P} f(pb_i(t))$
36: Até que o valor de f(*solução*) seja bom o suficiente ou $t = t_{max}$

Fonte: Montes de Oca et al (2009a).

O processo de alteração da topologia permite que nas iterações iniciais o algoritmo explore uma maior área do espaço de buscas e compartilhe rapidamente com as demais partículas informações sobre as regiões mais promissoras, uma vez que a topologia é completamente conectada a propagação da informação acontece rapidamente. À medida que as iterações avançam inicia-se o processo de remoção de arestas da vizinhança das partículas e a diminuição do peso de inércia. Tudo isto para garantir que as partículas permaneçam em suas regiões e não apresentem comportamento de fuga.

As particularidades do Frankenstein PSO o tornaram um algoritmo adaptável a execuções curtas ou longas apenas com o ajuste de alguns parâmetros de configuração (como o responsável pela atualização da topologia – parâmetro *k* no algoritmo do FPSO), por exemplo para execuções curtas temos: a adoção da topologia completamente conectada, que proporciona uma rápida propagação da informação entre as partículas; velocidade e peso de inércia altos nas iterações iniciais, que propiciam maior cobertura e deslocamento no espaço de buscas. Em execuções longas podemos retardar o processo de atualização da topologia ou definir que as remoções de arestas ocorram em um intervalo maior de iterações, uma vez que o método utilizado pelo FPSO para controle de topologia (AHPSO) permite este ajuste.

2.4 Frankenstein PSO com Convergência Garantida

A convergência garantida ou simplesmente *CG* é uma técnica bastante utilizada para evitar a convergência prematura do algoritmo a uma região não-ótima no espaço de soluções de acordo com Peer, Bergh e Engelbrecht (2003). Isto ocorre quando a posição atual de uma partícula coincide com sua melhor posição, pb_i, e a melhor visitada pelo enxame, gb_i. Assim o deslocamento da partícula fica à mercê do peso de inércia. Se este fato ocorrer nas iterações finais a partícula tende a permanecer na mesma região, uma vez que o peso de inércia possui valor próximo a zero o deslocamento da partícula seria mínimo. O estacionar da partícula provoca um movimento de convergência por parte do enxame a uma região sub-ótima.

A melhoria proposta pela implementação desta técnica corresponde a alterar a equação de atualização da velocidade para aquelas partículas que alcançaram a melhor posição visitada globalmente, *gbi*. Evitando-se assim uma indução a convergência prematura. Logo, uma nova equação de atualização da velocidade utilizada pelas partículas que atingirem a melhor posição global se faz necessária, a equação (9):

$$v_{i\,j}(t+1) = -x_{ij}(t) + gb_j(t) + wv_{ij}(t) + p(t)(1-2r_{ij}(t)) \quad (9)$$

em que o termo *p(t)* funciona como um raio de busca em relação ao melhor ponto visitado globalmente, *gb*. Para calcular *p(t)* utiliza-se a equação (10), disposta a seguir:

$$p(t+1) = \begin{cases} 2p(t), \text{ se } \#sucessos > s_c \\ 0{,}5p(t), \text{ se } \#falhas > f_c \\ p(t), \text{ caso contrário} \end{cases} \quad (10)$$

Sempre que a melhor área visitada globalmente, *gb*, for atualizada, ou seja, uma nova melhor posição é encontrada o contador de sucessos (#*sucessos*) é incrementado e a área do raio de busca tem seu valor dobrado. Quando o contador de falhas (#*falhas*) ultrapassa o valor estabelecido, f_c, o valor do raio de busca atribuído a ele é reduzido pela metade. Sempre que um contador tiver seu valor atualizado, #sucessos ou #falhas, o contador oposto terá seu valor zerado (PINGZHOU; ZHAOCAI, 2008).

2.5 Comentários Finais

Neste capítulo apresentamos os neurônios e as redes neurais artificiais, algumas funções de ativação e arquiteturas utilizadas neste trabalho, bem como a motivação biológica e, também, algumas limitações da técnica. Os pontos desfavoráveis ao uso das redes neurais artificiais deram origem a experimentações. Estas por sua vez tratam de mesclar diferentes métodos, sendo um deles uma rede neural, de forma que um seja capaz de suprir as deficiências do outro dando origem, assim, aos Sistemas Neurais Híbridos.

Métodos inspirados no comportamento animal, a exemplo da otimização por enxame de partículas – PSO - vem sendo utilizado em problemas de otimização numérica bem como em tarefas da área de aprendizado supervisionado em redes neurais artificiais. Este trabalho é inteiramente baseado em técnicas de otimização por enxame de partículas. A princípio apresentamos a técnica padrão e a variação adotada nesta pesquisa, o chamado Frankenstein PSO. Este algoritmo, apresentado pela primeira vez em 2009, é fruto de uma análise da combinação de sete diferentes variações do PSO, também explanados neste capítulo (*Constricted Particle Swarm Optimizer*; *Time-Decreasing Inertia Weight Variant*; *Increasing Inertia Weight Particle Swarm Optimization*; *Stochastic Inertia Weight Particle Swarm Optimization*; *Fully Informed Particle Swarm Optimizer*; *Self-organizing Hierarchical PSO With Time-Varying Acceleration Coefficients* e o *Adaptive Hierarchical Particle Swarm Optimizer*), tendo sido composto por apenas três deles (*Time-Decreasing Inertia Weight Variant*; *Fully Informed Particle Swarm Optimizer* e o *Adaptive Hierarchical Particle Swarm Optimizer*).

Neste capítulo também foi apresentado o *FPSO* com convergência garantida ou *FPSO:CG*. Técnica utilizada para evitar a convergência prematura do enxame a uma região sub-ótima no espaço de soluções. Este método consiste em estabelecer um raio de busca para a melhor posição visitada pelo enxame. Para isto são definidos dois contadores, #*sucessos* e #*fracasso*s, no qual toda vez que uma nova melhor posição visitada pelo enxame é encontrada o contador #*sucessos* é incrementado e o contador #*fracassos* é zerado, deste modo à área de busca associada ao “sucesso” é amplificada e a área associada ao “fracasso” é reduzida.

Capítulo 3

Treinamento de Redes Neurais com PSO

Está contido neste capítulo a definição da representação das soluções, a especificação das bases de dados utilizadas durante os experimentos, razões que motivaram a escolha da função de custo e como se deu o processo de treinamento das redes neurais pelo uso do Frankenstein PSO. Também estão especificados os critérios de configuração de cada algoritmo, bem como a validação estatística aplicada.

3.1 Introdução

O propósito do treinamento de redes neurais é delimitar a área de fronteira entre as classes do problema, em outras palavras, significa aumentar a capacidade de generalização. É dar a rede neural a habilidade de classificar corretamente padrões que não tenham sido apresentados anteriormente. Normalmente para treinar as redes neurais utilizam-se algoritmos locais específicos – a exemplo temos o *back-propagation*, *Levenberg-Marquardt*, *Resilient-Backpropagation*, *Quase-Newton* e outros – e/ou técnicas de busca global – Evolução Diferencial, utilizado por Zarth e Ludermir (2009), programação evolucionária por Yao e Liu (1997), Otimização por Enxame de Partículas em Carvalho e Ludermir (2006b, 2006c); Pingzhou e Zhaocai (2008); Van Wyk e Engelbrecht (2010), Algoritmos Genéticos como utilizado em Almeida e Ludermir (2006) – que utilizam o erro de treinamento como medida de avaliação. A aplicação de duas ou mais técnicas diferentes associadas a uma rede neural dá origem a Sistemas Neurais Híbridos, que geralmente combinam um algoritmo de busca global a algoritmos de busca local (*back-propagation*, *Levenberg-Marquardt* e outros).

Neste capítulo utilizaremos algoritmos híbridos no treinamento das redes neurais MLP. Estes algoritmos híbridos foram formados pelo *FPSO*, apresentado no capítulo

anterior (MONTES DE OCA et al, 2009a), associado a um algoritmo de busca local, *Levenberg-Marquardt* ou *Resilient-Backpropagation*.

O restante deste capítulo segue organizado da seguinte forma: a seção 3.2 aborda a representação das soluções; a seção 3.3 as funções de custo utilizadas; na seção 3.4 dispomos os experimentos realizados; seguido da seção de resultados e por fim as conclusões.

3.2 Representação das Soluções.

Para representar os pesos das conexões de uma rede neural MLP foram utilizados vetores de reais, para calcular o tamanho destes vetores foi preciso conhecer quantos seriam os pesos da rede, para isso foi utilizada a equação (11) a seguir:

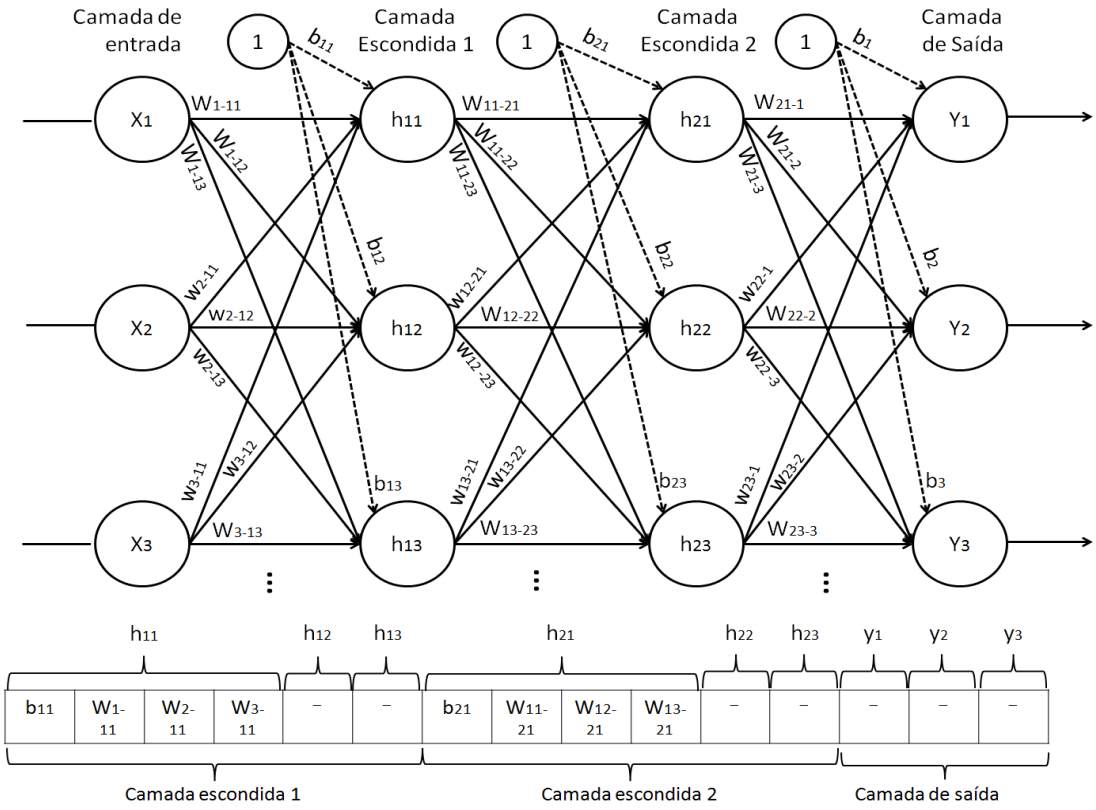
$$\text{Quantidade de Pesos} = (I + 1) \times H + (H + 1) \times O \qquad (11)$$

em que I é a quantidade de entradas do problema (input), H a quantidade de neurônios e O representa o número de saídas do problema (output). Os números 1 somados a I e H, entre os parênteses, representam o bias.

Neste trabalho foram adotadas redes neurais com arquitetura fixa, portanto *H* possui valor fixo e as conexões não sofrem poda, ou seja, a mesma quantidade de conexões da rede neural é mantida do início ao fim do processo de otimização (mantém-se a conexão máxima). Sendo algumas destas conexões fortalecidas, o peso atribuído a ela é incrementado enquanto que as outras são enfraquecidas, possuem valor baixo ou próximo a zero.

Na Figura 8 temos a representação gráfica dos pesos sinápticos e bias como parte do vetor de reais.

Figura 8: Pesos sinápticos e bias como componentes do vetor de reais.



Fonte: Carvalho (2007).

Para inicializar cada posição do vetor que representam as partículas foram gerados números aleatórios definidos dentro do intervalo [-2.0, +2.0] (quanto menor for o intervalo estabelecido menor será o espaço de busca para esta variável). Os parâmetros de velocidade e posição fizeram uso dos valores máximos permitidos, ou seja, $V_{max} = X_{max}$.

3.3 Função de Custo

Dentre as diversas funções de custo que podem ser empregadas (*Soma dos Erros Quadráticos – SSE*, *Raiz do Erro Quadrático Médio – RMS*, *Percentual do Erro Quadrático – SEP*, dentre outras) e suas inúmeras combinações, o *Erro Quadrático Médio Normalizado* (*NMSE*) foi a medida utilizada neste trabalho para avaliar a qualidade das soluções. Dentre as várias funções disponíveis a *NMSE* foi escolhida por ser uma medida de erro suave e também por ter sido a mesma adotada em

Carvalho (2007). O *NMSE* fornece uma medida do erro total cometido pela rede. Assim é possível saber o quanto a rede se distanciou do desempenho desejado, enquanto que o uso do *Erro Percentual de Classificação* (*CEP*), equação (13), deixa claro quantos padrões a rede classificou corretamente ou não. A fórmula para a medida de erro *NMSE* é disposta na equação (12).

O *NMSE* foi utilizado na fase de treinamento das redes neurais enquanto que o *Erro Percentual de Classificação* (*CEP*) foi utilizado na fase de testes. Estas foram as mesmas medidas adotadas em Carvalho (2007) para avaliar as soluções.

$$\text{NMSE} = (100/\text{NxC})\sum_{n=1}^{N}\sum_{k=1}^{C}(t_k^n - o_k^n) \tag{12}$$

$$\text{CEP} = 100\ x\ (\#\ \textit{erros de classificação})/\text{N} \tag{13}$$

nas equações (12) e (13) *N* representa o número de padrões do conjunto de dados; *C* o número de classes do problema; t_k^n a saída desejada para o padrão '*n*' e o_k^n a saída real obtida para o padrão '*n'*.

O critério de classificação mencionado na equação (12) foi calculado conforme a regra do *winner-takes-all ("o vencedor leva tudo"*), em que o neurônio da camada de saída que apresentar a mais alta taxa de ativação indica a classe que será atribuída ao padrão de entrada apresentado a rede.

3.4 Experimentos

A seguir será descrito o processo de treinamento das redes neurais MLP que fizeram uso das técnicas de otimização global *FPSO*. Na subseção 3.4.1 apresentamos as bases de dados utilizadas nos experimentos e na subseção 3.4.2 descrevemos a configuração adotada. Ao final, a subseção 3.4.3, o método estatístico empregado para comparar o desempenho dos algoritmos.

3.4.1 Base de Dados

Foram utilizadas sete bases de dados para problemas de classificação de padrões, sendo três da área médica, provenientes do proben1, Precheit (1994) e UCI, Frank e Asuncion (2010). A seguir temos uma breve descrição de cada base de dados.

- Câncer: Relacionada ao câncer de mama. Este conjunto de dados está dividido em duas partições, benigno e maligno, cuja classificação baseou-se na análise de dados microscópicos. A proporção entre as classes é a seguinte: 65,5% dos exemplos correspondem à classe benigna e 34,5% à classe maligna.
- Diabetes: relacionada à presença ou não do diabetes em índios Pima. Leva em conta dados pessoais e exames médicos. A proporção para os diabéticos é 65,1% e para os não diabéticos é de 34,9%.
- Coração: Classifica se um padrão apresenta ou não doenças do coração. Análise feita por meio do calibre de vasos sanguíneos.
- Vidros: esta base é resultado de uma análise química, de oito diferentes componentes, mais um índice de refração. Esta base foi criada para auxiliar análises forenses.
- Cavalos: Categorizam três possíveis diagnósticos veterinários – sobreviver, morrer ou ser sacrificado - para um cavalo com cólica. Para este conjunto de dados 62% dos exemplos correspondem à classe sobreviver, 24% são de cavalos que morreram e 14% para os que tiveram de ser sacrificados.
- Soja: Reconhece 19 diferentes doenças da soja. A análise leva em conta uma descrição do grão da soja, da planta e mais algumas informações relacionadas ao histórico da planta.
- Tireóide: Possui três classificações possíveis: hipotireoidismo, tireóide normal e hipertireoidismo. Esta base apresenta uma discrepância em relação à concentração de exemplos por classe, a hipotireoidismo conta com 5,1% dos padrões, a hipertireoidismo com 2,3% e a classe tireóide normal com 92,6%.

A quantidade exata de exemplos para cada um destes conjuntos pode ser conferida na Tabela 1.

Tabela 1: Caracterização e distribuição do número de padrões por base de dados.

Base de Dados	Entradas	Saídas	Nº exemplos de Treino	Nº exemplos de Validação	Nº exemplos de Teste
Câncer	9	2	350	174	175
Diabetes	8	2	384	192	192
Coração	35	2	460	230	230
Vidros	9	6	107	53	54
Cavalos	58	3	182	91	91
Soja	82	19	342	170	171
Tireóide	21	3	3600	1800	1800

Fonte: Autor (2011).

O número de neurônios na camada de entrada da rede neural (características do problema), ou seja, o *I* da equação (3.1) é representado pela coluna entradas. O número de classificações possíveis, *O* da equação (3.1), ou a quantidade de neurônios na camada de saída da rede neural é dado pela coluna saídas. Em relação à camada intermediária adotou-se uma arquitetura fixa, com seis neurônios, conforme mencionado na seção 3.2, representação das soluções. As sete bases de dados foram divididas em três subconjuntos seguindo a seguinte proporção: 50% dos dados representam o conjunto de treinamento; 25% o conjunto de validação e os outros 25% o conjunto de teste. O conjunto de validação foi utilizado para indicar o momento correto de parar o processo de treinamento.

3.4.2 Configurações

Para facilitar a identificação de cada algoritmo utilizado nos experimentos foram criadas siglas. Estas por sua vez foram formadas a partir da junção do FPSO mais o algoritmo de busca local utilizado no treinamento da rede neural. Então temos os seguintes: $FPSO_{Lm}$ (FPSO + *Levenberg-Marquardt*), $FPSO_{Rprop}$ (FPSO + *Resilient-backpropagation*) e $FPSO{:}CG_{Lm}$ (FPSO com convergência garantida + *Levenberg-Marquardt*). Estes três algoritmos foram comparados com os de busca local *Resilient-backpropagation* - Rprop (uma variação do algoritmo *back-propagation*, cujo propósito é acelerar a convergência do processo de treinamento. Ao invés de utilizar a taxa de

aprendizado o Rprop faz uso do sinal do gradiente do erro para indicar a direção do ajuste a ser feito nos pesos. Assim evita-se que sejam necessárias mais iterações para que o algoritmo alcance um ponto ótimo ou mesmo que assuma comportamento de estagnação em uma região plana na superfície de erro) e *Levenberg-Marquardt* – LM (ao contrário do algoritmo *back-propagation* que é baseado no gradiente descendente, o *Levenberg-Marquardt* utiliza taxa de aprendizado variável. Este algoritmo requer grande quantidade de memória e poder computacional).

A arquitetura das redes foi composta da seguinte maneira: camada de entrada – camada intermediária – camada de saída, sendo a camada intermediária composta por 6 neurônios, ou seja, foram utilizadas apenas redes neurais MLP com uma única camada escondida que continham 6 neurônios. Para compor o critério de classificação os dados de entrada da rede neural foram submetidos à regra do "*winner-takes-all"*, em que a unidade de saída que apresentar o maior valor determina a classe do padrão de entrada.

Tabela 2: Parâmetros de configuração dos algoritmos - PSO, CGPSO e CPSO-SK.

Algoritmo	Descrição	Valor
PSO	Tamanho do enxame	30 partículas
	Critério de parada	1000 iterações ou GL5[1]
	Critério de parada para decaimento de pesos	1000 iterações
	Medida de qualidade	NMSE
	Limite do espaço de busca	[-2.0, +2.0]
	Fatores de aceleração	c1 = c2 = 1.4960
	Peso de inércia	0,7298
CGPSO	P (raio) inicial	1
	Limiar de #*sucesso* e #*fracasso*	5
CPSO-SK	Fator de particionamento	k ≅ 1.3 x $\sqrt{pesos}$

Fonte: Carvalho (2007).

[1] GL_5 é o mesmo que *Critério de Parada Antecipado*, utilizado no conjunto de validação para estimar quando a rede começa a memorizar as nuances dos dados de treino. O treinamento da rede, então, é interrompido quando o erro de validação, GL_5, atinge 5%.

Os parâmetros de configuração utilizados pelos algoritmos referenciados, (CARVALHO, 2007) estão descritos na Tabela 2, enquanto que na Tabela 3 é possível verificar as configurações adotadas para os experimentos com os algoritmos $FPSO_{Lm}$, $FPSO_{Rprop}$ e $FPSO:CG_{Lm}$. Na Tabela 4 temos a descrição dos parâmetros utilizados pela rede neural.

Em Carvalho (2007) CGPSO significa PSO com convergência garantida. Este algoritmo utiliza os mesmos parâmetros que o PSO, no entanto possui dois parâmetros adicionais (raio inicial e os limiares de sucesso e fracasso). CPSO-SK é o mesmo que PSO cooperativo e também utiliza os mesmos parâmetros do PSO, no entanto possui um parâmetro adicional – o fator de particionamento, no caso *k*.

Os valores adotados para o desenvolvimento deste trabalho levaram em consideração as configurações utilizadas em Carvalho (2007), disponíveis na Tabela 1, a exceção foi o número de iterações para o critério de parada. No presente trabalho o número de iterações foi 10 vezes menor do que em Carvalho (2007), devido a restrições de tempo de execução.

Tabela 3: Parâmetros de configuração dos algoritmos $FPSO_{Lm}$, $FPSO_{Rprop}$ e $FPSO:CG_{Lm}$.

Algoritmo	Descrição	Valor
FPSO	Tamanho do enxame	30 partículas
	Critério de parada	100 iterações ou GL_5
	Medida de qualidade	NMSE
	Limite do espaço de busca	[-2.0, +2.0]
	Fator de inércia	[0.9 a 0.4]
	K	30
FPSO:CG	P (raio) inicial	1
	Limiar de #*sucesso* e #*fracasso*	5

Fonte: Autor (2011).

A medida de qualidade NMSE, assim como em Carvalho (2007), foi empregada apenas no treinamento das partículas enquanto que na fase de testes foi utilizado o CEP.

Tabela 4: Parâmetros de configuração da rede neural.

Descrição da Rede Neural	Valor
Nº de neurônios escondidos	6
Número máximo de iterações	100
Funções de ativação	Tangente sigmóide – Linear
Algoritmo de treino	Levenberg-marquardt
Nº de falhas de validação	5

Fonte: Autor (2011).

3.4.3 Comparação dos Algoritmos

Para obtenção das médias CEP os algoritmos foram submetidos a 30 execuções independentes cada um. A cada nova execução os subconjuntos de dados – treinamento, validação e teste – foram divididos de forma aleatória seguindo as proporções indicadas na subseção 3.4.1. Ao final de cada execução foram armazenados os erros de classificação para os dados de teste, calculado conforme a equação (13).

Para validar estatisticamente os resultados obtidos, as médias finais de $FPSO_{Lm}$, $FPSO_{Rprop}$ e $FPSO{:}CG_{Lm}$ foram submetidas ao seguinte teste de hipótese, sendo adotado α=5% ou nível de confiança de 95% (WAYNE, 1990):

$$t = \frac{\bar{x}_1 - \bar{x}_2}{\sqrt{\frac{S_1^2}{n_1} + \frac{S_2^2}{n_2}}} \qquad (14)$$

em que $\bar{x}$ representa a média, S^2a variância e *n* o número de execuções independentes para a amostra em questão.

Os testes estatísticos foram aplicados comparando os algoritmos $FPSO_{Lm}$, $FPSO_{Rprop}$, $FPSO{:}CG_{Lm}$ aos algoritmos de busca local LM e Rprop aplicados as mesmas bases de dados. Primeiro foi realizado o teste bilateral para verificar se a média dos resultados dos algoritmos era estatisticamente diferente. Se o resultado fosse diferente, era, então, aplicado o teste unilateral à esquerda para verificar se a média

dos resultados de um algoritmo era estatisticamente menor que a média dos resultados do segundo algoritmo.

3.5 Resultados

Os resultados dos experimentos são apresentados na Tabela 5 que contém as médias (μ) e o desvio padrão (σ) do erro percentual de classificação (CEP) para cada base de dados em cada algoritmo. Os valores destacados em negrito correspondem às menores médias de erro por base de dados.

As siglas Lm e Rprop associados aos FPSOs na tabela acima indicam o algoritmo de busca local empregado no treinamento da rede neural. Estes representam o *Levenberg-Marquardt* e o *Resilient-backpropagation* respectivamente.

Tabela 5: Media e desvio padrão do erro percentual de classificação para os algoritmos $FPSO_{LM}$, $FPSO_{Rprop}$, LM, Rprop e $FPSO:CG_{Lm}$ utilizados no treinamento de redes neurais.

Base de Dados	$FPSO_{Lm}$		$FPSO_{Rprop}$		LM		Rprop		$FPSO:CG_{Lm}$	
	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
Câncer	**2,969**	1,688	3,352	1,484	4,705	1,780	4,076	1,774	3,886	1,337
Diabetes	22,708	2,932	22,778	3,595	26,163	3,508	24,080	3,057	**21,979**	4,711
Coração	**18,580**	3,319	21,058	6,023	21,986	3,237	19,710	2,985	19,159	3,189
Vidro	**35,598**	9,160	39,748	10,821	43,270	8,050	47,044	9,971	36,855	9,133
Cavalos	**34,762**	5,174	36,557	5,588	43,150	5,242	37,106	4,869	35,018	5,813
Soja	**38,157**	3,542	63,157	7,150	40,510	4,621	61,706	10,542	41,784	7,749
Tireóide	**1,774**	0,398	5,978	0,504	3,556	1,203	6,337	0,822	4,526	1,923

Fonte: Autor (2011).

O tempo médio das execuções, em segundos, para cada algoritmo e base de dados está registrado na Tabela 6.

Tabela 6: Média e desvio padrão do tempo de execução, em segundos, para os algoritmos $FPSO_{Lm}$, $FPSO_{Rprop}$, LM, Rprop e $FPSO:CG_{Lm}$ utilizados no treinamento de redes neurais.

Base de Dados	$FPSO_{Lm}$		$FPSO_{Rprop}$		LM		Rprop		$FPSO:CG_{Lm}$	
	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
Câncer	1463,69	9,92	1763,14	66,66	0,43	0,12	0,46	0,15	870,78	670,54
Diabetes	1504,08	8,60	1516,54	21,93	0,37	0,06	0,38	0,08	2012,04	569,09
Coração	2391,57	30,73	907,33	8,84	0,86	0,43	0,35	0,08	2321,17	85,96
Vidro	1807,52	44,99	875,68	111,62	0,67	0,18	0,79	0,82	1736,22	50,60
Cavalos	2749,33	41,17	1289,85	11,67	0,86	0,22	0,53	0,14	2348,60	45,57
Soja	21226,23	4071,07	1613,60	25,31	8,09	1,64	1,76	1,09	17052,38	2203,70
Tireóide	11185,60	946,62	1486,32	79,98	5,87	4,91	3,95	1,93	7882,60	1179,19

Fonte: Autor (2011).

O tempo de execução dos algoritmos híbridos notadamente são muito maiores aos de busca local, *Levenberg-Marquardt* e *Resilient-Back-propagation*, entretanto o desempenho alcançado pelos que fazem uso da técnica FPSO consorciado a um algoritmo de busca local é compensado quando avaliamos as médias de erros obtidas.

Os tempos de execução dos algoritmos FPSO são elevados pelo fato de que para cada componente *i* do vetor de partículas e para cada iteração são computadas as quantidades de vizinhas topológicas e novos fatores de constrição, além do laço para retirada das arestas e redefinição do peso de inércia.

Os algoritmos da Tabela 6 foram comparados e os resultados dos testes de hipótese por base de dados podem ser visualizados na Tabela 7. Apenas os resultados relevantes estatisticamente foram relacionados, ou seja, apenas os resultados cujo algoritmo 1 foi melhor que o algoritmo 2.

Tabela 7: Resultado dos testes de hipótese para as sete bases de dados que utilizaram algoritmo híbrido no treinamento de redes neurais MLP[2].

Base de Dados	Algoritmo 1	Algoritmo 2	Valor Obtido
Câncer	$FPSO_{Lm}$	LM	-3,8761
	$FPSO_{Lm}$	Rprop	-2,7857
	$FPSO_{Lm}$	FPSO: GC_{Lm}	-2,3325
Diabetes	$FPSO_{Lm}$	LM	-4,1391
Coração	$FPSO_{Lm}$	LM	-4,0239
Vidro	$FPSO_{Lm}$	LM	-3,4459
	$FPSO_{Lm}$	Rprop	-4,6302
Cavalos	$FPSO_{Lm}$	LM	-6,2377
Soja	$FPSO_{Lm}$	LM	-2,2135
	$FPSO_{Lm}$	Rprop	-11,5980
	$FPSO_{Lm}$	FPSO: GC_{Lm}	-2,3316
	$FPSO_{Lm}$	$FPSO_{Rprop}$	-17,1609
Tireóide	$FPSO_{Lm}$	LM	-7,7028
	$FPSO_{Lm}$	Rprop	-27,3656
	$FPSO_{Lm}$	FPSO: GC_{Lm}	-7,6758
	$FPSO_{Lm}$	$FPSO_{Rprop}$	-35,8553

Fonte: Autor (2011).

Na Tabela 8 são apresentados a média e desvio padrão CEP dos algoritmos utilizados em (Carvalho (2007) para as bases de dados câncer, diabetes e coração em comparação com o desempenho do algoritmo $FPSO_{Lm}$, cuja média de erro obtida foi, na maioria das vezes, a menor dentre as três versões apresentadas neste trabalho ($FPSO_{Lm}$, $FPSO_{Rrop}$ e $FPSO{:}CG_{Lm}$) para ajuste dos pesos de uma rede neural MLP.

[2] O algoritmo 1 adotado na comparação com os demais foi o $FPSO_{Lm}$ por este ter alcançado as menores médias de erros, conforme a Tabela 6.

Tabela 8: Média e desvio padrão do CEP para cada algoritmo proposto no trabalho base em relação ao algoritmo FPSOLm.

Base de Dados	PSO-GL$_5$ (CARVALHO, 2007)		PSO-WD (CARVALHO, 2007)		GCPSO-WD (CARVALHO, 2007)		FPSO$_{Lm}$	
	μ	σ	μ	σ	μ	σ	μ	σ
Câncer	3,542	1,182	3,440	1,425	3,805	1,063	**2,969**	1,688
Diabetes	24,687	2,345	23,708	2,606	**22,677**	2,566	22,708	2,932
Coração	20,547	2,379	17,904	2,108	**17,356**	1,963	18,580	3,319

Fonte: Autor (2011).

A Tabela 9 mostra que o algoritmo FPSO$_{Lm}$ apresentou melhor desempenho estatístico que o algoritmo PSO-GL5 apenas nas bases de dados diabetes e coração e ao GCPSO-WD na base câncer. No entanto é preciso salientar que o FPSO$_{Lm}$ contou com 10 vezes menos execuções – de acordo com a Tabela 3 se comparado com a Tabela 2 - durante o treinamento da rede neural e mesmo assim conseguiu alcançar bons resultados. Para as demais configurações o FPSO$_{Lm}$ mostrou-se equivalente estatisticamente aos demais.

Tabela 9: Resultado dos testes de hipótese entre FPSO$_{LM}$ e os algoritmos de ajuste dos pesos apresentados em Carvalho (2007).

Base de Dados	Algoritmo 1	Algoritmo 2	Valor Obtido
Câncer	FPSO$_{Lm}$	GCPSO-WD	-2,4381
Diabetes	FPSO$_{Lm}$	PSO-GL5	-3,1427
Coração	FPSO$_{Lm}$	PSO-GL5	-2,8380

Fonte: Autor (2011).

3.6 Conclusão

Neste capítulo foram apresentados os aspectos gerais do uso de três algoritmos híbridos formados a partir de uma variação da otimização por enxame de partículas, FPSO, com algoritmos de busca local (*Levenberg-Marquardt* e *Resilient-Back-propagation*), foram eles: FPSO$_{Lm}$ e FPSO$_{Rprop}$ e uma versão híbrida que utilizou a técnica da convergência garantida, FPSO:CG$_{Lm}$, no processo de ajuste dos pesos das

conexões de redes neurais MLP do tipo *feed-forward*. O modo de representação das soluções, as bases de dados utilizadas, as configurações adotadas, bem como o teste estatístico aplicado para validar os experimentos.

Nesta etapa do trabalho a comparação dos resultados experimentais realizados entre os algoritmos $FPSO_{Lm}$, $FPSO_{Rprop}$ e $FPSO{:}CG_{Lm}$ demonstraram que o uso da técnica de convergência garantida não apresentou melhoria significativa nos resultados. Apesar de o $FPSO{:}CG_{Lm}$ ter alcançado a menor média de erros em uma das bases de dados utilizada, diabetes, o teste de hipótese não comprovou que este algoritmo foi estatisticamente melhor que os demais. Por este motivo a técnica de convergência garantida não foi empregada no processo simultâneo de otimização dos pesos e arquiteturas, descrito no capítulo a seguir. Uma equivalência estatística foi comprovada entre os algoritmos $FPSO_{Lm}$ e $FPSO_{Rprop}$. No entanto foi utilizado apenas a versão que fez uso do algoritmo de busca local Levenberg-Marquardt, $FPSO_{Lm}$, por este ter obtido as menores médias de erro na maioria das bases de dados testadas.

Apesar de os testes estatísticos comprovarem que o $FPSO_{Lm,}$ que obteve as menores médias de erros na maioria das bases de dados foi superior a apenas 2 variações de PSO propostas em Carvalho (2007) é importante observar que o $FPSO_{Lm}$ contou com dez vezes menos execuções em relação aos algoritmos apresentados em Carvalho (2007).

O tempo de execução dos algoritmos híbridos propostos nesta pesquisa, $FPSO_{Lm}$, $FPSO_{Rprop}$ e $FPSO{:}CG_{Lm}$, foi um dos limitantes do processo de otimização das redes neurais. Se imaginarmos uma projeção dos resultados no qual fosse possível adicionarmos dez vezes mais execuções, poderíamos afirmar que os métodos propostos seriam, sim, melhores que aqueles apresentados em Carvalho (2007). Por este motivo devemos continuar estudando formas de diminuir o tempo de execução destes algoritmos, seja adotando medidas como computação paralela ou diminuir o tamanho da população de partículas ao longo das iterações, mantendo apenas aquelas que alcançaram melhores posições no espaço de busca. Esta medida evitaria que os algoritmos híbridos propostos investissem tempo e poder computacional em partículas que não proporcionam bons resultados. Este tempo então poderia ser

revertido em novas iterações de avaliação para aquelas localizadas em regiões do espaço de busca mais promissor.

Capítulo 4

Ajuste Simultâneo de Pesos e Arquiteturas com FPSO

Este capítulo, em adição ao anterior - ajuste dos pesos das conexões, quando no treinamento da rede neural - estabelece a redefinição das arquiteturas. Esta combinação, arquiteturas e pesos da rede neural, se dá de modo simultâneo, ou seja, à medida que uma melhor arquitetura é definida os pesos das conexões da rede neural são recalibrados. Neste processo foram utilizados o algoritmo do Frankenstein PSO, as mesmas bases de dados e resultados do capítulo anterior.

4.1 Introdução

O processo de treinamento de redes neurais com ajuste simultâneo de arquiteturas e pesos diferencia-se do mencionado no Capítulo 3 pelo fato de, agora, o processo contemplar os dois passos o de estabelecer a arquitetura adequada e ajustar os pesos das conexões recém estabelecidas. Na primeira etapa define-se a quantidade de camadas intermediárias bem como a quantidade de neurônios ou nodos que cada uma possuirá. Em um segundo momento são calibrados os pesos das conexões definidos pela primeira etapa.

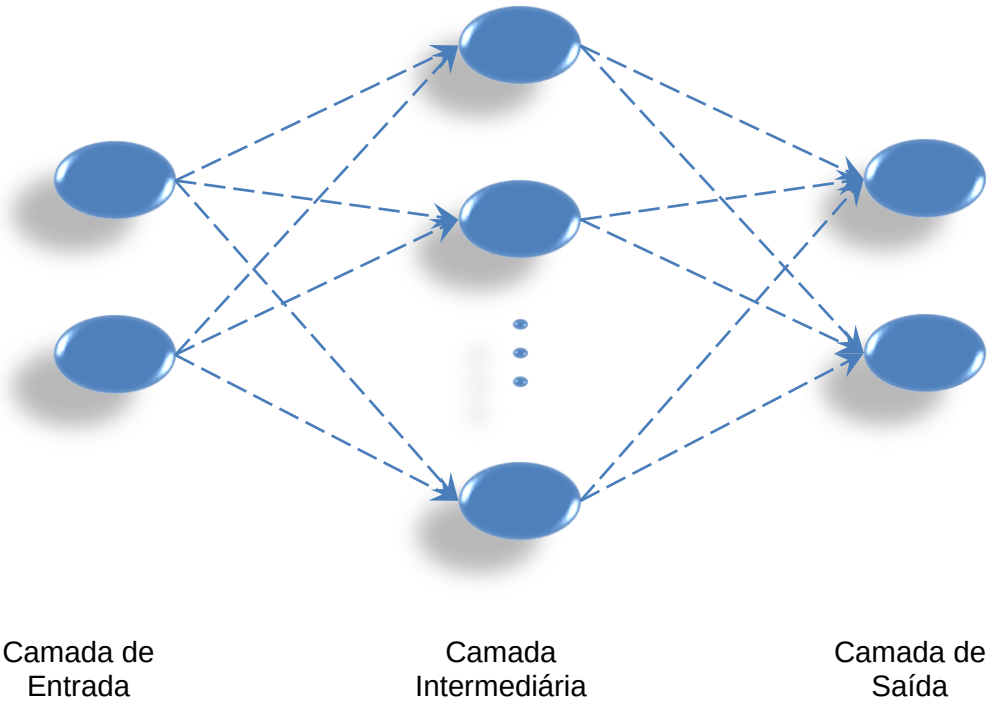
A definição da arquitetura de uma rede neural não é uma tarefa fácil, pois quando a rede possui parâmetros além do necessário ocorre *overfitting* (VAN WYK; ENGELBRECHT, 2010) (significa que a rede memoriza os dados de treinamento e quando é apresentada a dados desconhecidos – dados de teste – produz altas taxas de erro) e quando possui parâmetros a menos ocorre o inverso, *underfitting* (a rede não possui conexões suficientes, com isso apenas parte das características do problema são aprendidas. O que também causa alta taxa de erro).

Normalmente esta parte do processo de treinamento é realizada na base da tentativa e erro ou através da consulta a um especialista. Por este motivo são cada vez mais

frequentes o desenvolvimento e aplicação de técnicas computacionais com o propósito de definir automaticamente a arquitetura da rede. O objetivo destes métodos é fixar redes neurais de baixa complexidade (a complexidade é dada pelo tamanho do vetor de pesos, quanto maior o vetor maior é a complexidade da rede) que produzam erros dentro de uma faixa mínima aceitável.

Neste capítulo apresentamos uma metodologia automática para otimização de arquiteturas e ajuste dos pesos de redes neurais MLP do tipo *feed-forward* fundamentadas no algoritmo de otimização por enxame de partículas FPSO. Esta metodologia faz uso dos resultados obtidos no capítulo 3 e agora além do ajuste dos pesos também realiza a definição do número de neurônios na única camada escondida considerada – assim como em Carvalho (2007). Neste trabalho consideramos redes neurais de arquitetura fixa, cujo único parâmetro variável é a quantidade de neurônios na única camada intermediária, ou seja, as arquiteturas são do tipo: camada de entrada – uma camada intermediária – camada de saída, de acordo com a Figura 9.

Figura 9: Esquema de uma rede neural MLP do tipo *feed-forward*.



Fonte: Autor (2011).

Este capítulo segue organizado da seguinte forma: na seção 4.2 são apresentados os aspectos gerais relativos à otimização simultânea dos pesos e arquiteturas; na seção seguinte, a 4.3, são detalhados os experimentos realizados nas 7 bases de dados. Na seção 4.4 dispomos os resultados obtidos e por fim temos as conclusões na seção 4.5.

4.2 Otimização dos pesos e arquiteturas com PSO

A abordagem utilizada neste trabalho é similar à apresentada em Carvalho (2007), em que dois algoritmos são executados simultaneamente, sendo um para definir o número de neurônios na única camada intermediária e o outro para ajustar os pesos das conexões da rede neural. Os dois enxames (o de arquitetura e o de pesos) utilizaram diferentes partições do conjunto de dados. A proporção dos subconjuntos de dados foram as mesmas da subseção 3.4.1. O conjunto de treinamento foi utilizado pelo algoritmo responsável pelo ajuste dos pesos; o conjunto de validação pelo algoritmo responsável pela definição da arquitetura e o conjunto de teste utilizado no final do processo para avaliar a melhor configuração encontrada.

Os algoritmos desenvolvidos neste trabalho seguiram a mesma metodologia que a apresentada em Carvalho (2007) sendo o $FPSO_{Lm}$ sempre utilizado no ajuste dos pesos enquanto que na definição da arquitetura foram utilizados o PSO e o FPSO. Portanto os novos algoritmos desenvolvidos foram: PSO-$FPSO_{Lm}$ e FPSO-$FPSO_{Lm}$, em que o primeiro termo indica o algoritmo utilizado para definir as arquiteturas e o segundo representa o algoritmo utilizado para o treinamento da rede neural.

4.2.1 Representação das Soluções

O algoritmo responsável pela definição da arquitetura (PSO ou FPSO) possui uma estrutura de dados diferenciada. Nesta estrutura são armazenados dois valores, além das configurações inerentes ao algoritmo em questão (PSO ou FPSO), são eles: um número natural – para representar o número de neurônios na camada escondida – e um vetor de reais para armazenar os melhores pesos sinápticos encontrados até o momento. Este vetor com os melhores pesos sinápticos é referenciado aqui por *net*.

O algoritmo que calibra os pesos das conexões utiliza a mesma abordagem descrita na subseção 3.2, ou seja, utiliza vetores de reais onde cada componente representa um peso da rede neural.

4.2.2 Algoritmo de otimização das arquiteturas e pesos

O método de otimização utilizado neste trabalho consiste na execução simultânea de dois algoritmos de otimização por enxame de partículas, sendo um para a definição da arquitetura, baseado no erro do conjunto de validação, e o outro para ajuste dos pesos sinápticos – baseado no erro do conjunto de treinamento. A seguir o pseudo-código para a otimização das arquiteturas e pesos.

Algoritmo 3: Pseudo-código da otimização para a definição das arquiteturas e ajuste dos pesos de uma rede neural MLP.

```
1:  Inicializar randomicamente a população de arquiteturas A,
2:  Repita
3:      Para cada particular Ai da população A faça
4:          Iniciar Pi
5:          Inserir Ai.net em Pi
6:          Executar Pi por t iterações utilizando o conjunto de treinamento
7:          Ai.net = Pi.ŷ
8:          Avaliar f(Ai.net) utilizando o conjunto de validação
9:      Fim do para
10:     Para cada partícula Ai da população A faça
11:         Atualize velocidade e posição de Ai de acordo com a equação
            (4) e equação (2).
12:         Atualize ai.net para a nova arquitetura representada por Ai
13      Fim do para
14: Até que o critério de parada seja satisfeita (por exemplo: nº máximo de
    execução ou taxa de erro mínimo alcançado).
```

Fonte: Carvalho (2007).

Para cada arquitetura considerada é iniciado um processo de otimização dos pesos. Sempre uma das partículas é iniciada com o melhor vetor de pesos sinápticos

encontrados até o momento (vetor *net* – definido pela partícula de arquitetura). Quando o processo de otimização dos pesos estiver sido concluído, o melhor vetor encontrado até o momento é armazenado de volta na partícula de arquitetura, *net*.

O processo de otimização simultânea das arquiteturas e pesos é muito mais lento se comparado apenas com o ajuste dos pesos. Por isso adotamos a seguinte configuração em relação à adotada em Carvalho (2007).

Tabela 10: Configurações dos algoritmos de otimização simultânea das arquiteturas e pesos em relação ao trabalho base.

Configuração	Presente Trabalho	(CARVALHO, 2007)
Nº de partículas do algoritmo de arquitetura	20	20
Nº de iterações do algoritmo de arquitetura	10	15
Nº de iterações do algoritmo de pesos	100	100

Fonte: Autor (2011).

É possível observar que na Tabela 10 apenas o número de iterações do algoritmo de otimização por enxame de partículas responsável pela definição das arquiteturas possui valor diferente ao utilizado no trabalho base. Este valor foi reduzido para 10 devido ao tempo total de execução do algoritmo de otimização das arquiteturas e pesos.

4.3 Experimentos

Nos experimentos realizados com a metodologia simultânea de otimização das arquiteturas e pesos foram comparados os desempenhos dos algoritmos PSO-FPSO$_{Lm}$ e FPSO-FPSO$_{LM}$ com outros presentes na literatura. Foram utilizadas nos experimentos as mesmas 7 (sete) bases de dados relacionadas na subseção 3.4.1 e a mesma estratégia de validação estatística.

Os parâmetros de configuração dos algoritmos PSO-FPSO$_{Lm}$ e FPSO-FPSO$_{Lm}$ estão descritos na Tabela 11.

Tabela 11: Parâmetros de configuração dos algoritmos de ajuste simultâneo dos pesos e arquiteturas.

ARQUITETURAS	
PSO	
Tamanho do enxame	20
Número de iterações de avaliação da arquitetura	10
Medida de Qualidade	NMSE (conj. de validação)
Limite do espaço de busca	[1, 12]
Coeficiente de aceleração	1,4960
Peso de inércia	0,7298
FPSO	
Tamanho do enxame	20
Número de iterações de avaliação da arquitetura	10
Medida de Qualidade	NMSE (conj. de validação)
Limite do espaço de busca	[1, 12]
Peso de inércia (ω)	0,9 a 0,4, linearmente decrescente
K	20
PESOS	
FPSO	
Tamanho do enxame	30
Número de iterações de avaliação dos pesos	100
Algoritmo de busca local	Levenberg-Marquardt (Lm)
Medida de Qualidade	NMSE (conj. de treino)
Limite do espaço de busca	[-2.0, 2.0]
Peso de inércia (ω)	0,9 a 0,4, linearmente decrescente
K	30

Fonte: Autor (2011).

Assim como no ajuste dos pesos sinápticos, Capítulo 3, em que as redes neurais possuíam arquitetura fixa – 6 neurônios na camada escondida - o processo de ajuste simultâneo da arquitetura e pesos também assumiu redes neurais com arquitetura fixa – apenas uma camada intermediária, sendo, no entanto 12 o número máximo de

nodos na camada escondida. O critério de classificação também foi o mesmo adotado no capítulo anterior, *winner-takes-all*, ou seja, o vencedor leva tudo.

4.4 Resultados

Na Tabela 12 temos os resultados experimentais dos algoritmos PSO-FPSO$_{Lm}$ e FPSO-FPSO$_{Lm}$ em que estão dispostas as médias (μ) e desvios padrão (σ) dos erros percentuais de classificação (CEP) para cada algoritmo em cada uma das bases de dados. Em negrito destacamos a menor média de erro alcançada em cada base.

Tabela 12: Média e desvio padrão do erro percentual de classificação para os algoritmos PSO-FPSO$_{Lm}$ e FPSO-FPSO$_{Lm}$.

Base de dados	PSO-FPSO$_{Lm}$		FPSO-FPSO$_{Lm}$	
	μ	σ	μ	σ
Câncer	3,676	1,761	**3,333**	1,334
Diabetes	22,205	4,183	**21,849**	4,173
Coração	19,015	2,866	**17,136**	4,518
Vidros	39,436	6,955	**36,118**	4,836
Cavalos	35,092	5,395	**33,603**	5,828
Soja	58,2661	4,300	**39,467**	5,985
Tireóide	3,715	2,542	**2,715**	1,557

Fonte: Autor (2011).

O tempo das execuções segue disposto em segundos (s) na Tabela 13. Se compararmos a Tabela 6 com a Tabela 13 podemos perceber a grande diferença entre o tempo médio das execuções. Isto se dá pelo fato de que a primeira ajusta apenas os pesos das conexões da rede neural enquanto que na última para cada partícula de arquitetura existem outras 30 partículas para ajuste dos pesos.

Tabela 13: Média e desvio padrão do tempo de execução em segundos para os algoritmos PSO-FPSO$_{Lm}$ e FPSO-FPSO$_{Lm}$.

Base de dados	PSO-FPSO$_{Lm}$		FPSO-FPSO$_{Lm}$	
	μ	σ	μ	σ
Câncer	3335,308	924,725	59068,385	33948,658
Diabetes	3521,203	474,382	117453,378	18715,534
Coração	5848,261	2050,429	132743,420	19116,983
Vidros	16957,091	2274,115	84166,164	57430,114
Cavalos	8178,1218	4015,4315	37583,445	79219,158
Soja	121769,5872	12957,6078	154797,125	87358,3498
Tireóide	216468,636	31404,085	238144,236	105228,3308

Fonte: Autor (2011).

Os resultados experimentais demonstram que o algoritmo FPSO-FPSO$_{Lm}$ obteve os melhores resultados em todas as sete bases de dados testadas (ver Tabela 13). Entretanto para afirmar que a utilização do algoritmo FPSO-FPSO$_{Lm}$ obteve, de fato, o melhor desempenho ele foi comparado ao PSO-FPSO$_{Lm}$ - desenvolvido neste trabalho - e a outros algoritmos encontrados na literatura - PSO-PSO:WD, PSO-GCPSO:WD, PSO-GCPSO:GL5 e GaTSa.

Os três primeiros algoritmos foram desenvolvidos pelo mesmo autor e estão definidos em Carvalho e Ludermir (2007). Trata-se da aplicação do PSO padrão para a definição das arquiteturas e uso de três diferentes variações do PSO ao ajuste dos pesos. Estas combinações foram: PSO:WD – que é o PSO padrão combinado a técnica de decaimento de pesos; GCPSO:WD – PSO com convergência garantida associado a técnica de decaimento de pesos e GCPSO:GL5 – PSO com convergência garantida associado a contagem de erros do conjunto de validação (GL5).

Na Tabela 14 dispomos as médias e desvios-padrão CEP obtidos por outros trabalhos encontrados na literatura que realizam o ajuste dos pesos e definição da arquitetura de redes neurais MLP. Podemos observar que o algoritmo apresentado neste trabalho, FPSO-FPSO$_{Lm}$, obteve as menores médias de erros.

Tabela 14: Média e desvio padrão dos algoritmos PSO-FPSO$_{Lm}$ e FPSO-FPSO$_{Lm}$ em relação a outros trabalhos presentes na literatura que propuseram a definição das arquiteturas e ajuste dos pesos de uma rede neural MLP.

		Câncer	Diabetes	Coração	Vidros	Cavalos	Soja	Tireóide
PSO-FPSO$_{Lm}$	μ	3,676	22,205	19,015	39,436	35,092	58,266	3,715
	σ	1,761	4,183	2,866	6,955	5,395	4,300	2,542
FPSO-FPSO$_{Lm}$	μ	**3,333**	**21,849**	**17,136**	**36,118**	**33,603**	**39,468**	**2,715**
	σ	1,334	4,173	4,518	4,836	5,828	5,985	1,557
PSO-GCPSO:GL5 (CARVALHO; LUDERMIR, 2007)	μ	4,754	24,906	19,383	-	-	-	-
	σ	4,427	3,529	2,268	-	-	-	-
PSO-PSO:WD (CARVALHO; LUDERMIR, 2007)	μ	4,137	23,541	24,906	-	-	-	-
	σ	1,506	3,159	3,529	-	-	-	-
PSO-GCPSO:WD (CARVALHO; LUDERMIR, 2007)	μ	4,560	23,604	19,383	-	-	-	-
	σ	1,461	3,013	2,268	-	-	-	-
GaTSa (ZANCHETTIN; LUDERMIR; ALMEIDA, 2011)	μ	7,192	27,062	-	55,143	38,700	62,941	7,151
	σ	4,031	3,109	-	6,082	1,585	5,679	0,890

Fonte: Autor (2011).

O algoritmo GaTSa apresentado em Zanchettin e Ludermir (2009) também foi desenvolvido para definir a arquitetura e ajustar os pesos de uma rede neural MLP. O GaTSa é um algoritmo híbrido composto por Algoritmo Genético, *Tabu Search*, *Simulated Annealing* e o algoritmo de busca local *back-propagation*. Este algoritmo possui duas fases bem definidas. A primeira consiste de uma busca global, na qual novas soluções são geradas, esta capacidade foi herdada do Algoritmo Genético, bem como o uso de memória, característica do *Tabu Search*. Em um segundo momento o

algoritmo entra na fase de busca local. Nesta hora o GaTSa faz uso de características provenientes do *back-propagation*, que proporciona uma solução mais precisa.

De forma geral podemos dizer que o GaTSa funciona da seguinte maneira: Uma arquitetura com tamanho mínimo é definida como a solução inicial; a partir daí novas soluções são geradas - como em um algoritmo genético. O custo de cada nova solução é avaliado e a melhor delas é escolhida como no *Tabu Search*. Entretanto esta solução poderá ou não ser aceita, o critério de aceitação é o mesmo que o utilizado no *Simulated Annealing* – se a nova solução tiver um custo menor ela é aceita, senão poderá ser rejeitada conforme o cálculo da probabilidade. Soluções previamente encontradas são marcadas como tabu – assim como no *Tabu Search*. Durante a busca por novas soluções o tamanho do cromossomo é aumentado a fim de encontrar a melhor solução conforme os critérios de aceitação. Ao final do processo apenas a melhor solução é retornada.

4.5 Conclusão

Neste capítulo apresentamos o método de otimização das arquiteturas e pesos sinápticos para redes neurais MLP, do tipo *feed-forward* que fizeram uso de uma nova variação publicada recentemente da otimização por enxame de partículas, chamado Frankenstein PSO ou FPSO (MONTES DE OCA et al, 2009a).

A metodologia empregada no desenvolvimento dos dois novos algoritmos - PSO-FPSO$_{Lm}$ e FPSO-FPSO$_{Lm}$ - foi a mesma disposta em Carvalho (2007). Na qual o autor utilizou de forma alternada dois algoritmos PSO, um para definir a arquitetura e outro para calibrar os pesos das conexões da rede neural MLP.

O algoritmo FPSO-FPSO$_{Lm}$ foi superior ao algoritmo PSO-FPSO$_{Lm}$ em termos da média do erro percentual de classificação para todas as bases de dados investigadas nesse trabalho. Contudo, como esperado, o algoritmo FPSO-FPSO$_{Lm}$ foi inferior ao algoritmo PSO-FPSO$_{Lm}$ em termos da média de tempo de execução.

Em comparação com os algoritmos da literatura PSO-GCPSO:GL5, PSO-PSO:WD e PSO-GCPSO:WD disponíveis em Carvalho e Ludermir (2007) e GaTSa em Zanchettin

e Ludermir (2009), o algoritmo FPSO-FPSO$_{Lm}$ é claramente melhor do que esses algoritmos em termos da média do erro percentual de classificação para as bases de dados adotadas nessa dissertação. Os resultados referentes a este capítulo encontram-se publicados em Lima e Ludermir (2011).

Capítulo 5

Enxames Heterogêneos

Neste capítulo há uma breve introdução as possíveis classificações quanto a heterogeneidade do enxame. Os parâmetros de configuração adotados, base de dados utilizadas, disposição do processo de treinamento da rede neural e resultados obtidos são também apresentados.

5.1 Introdução

Os modelos de otimização por enxame de partículas discutidos até o momento neste trabalho – PSO padrão e FPSO – são ditos homogêneos porque todas as partículas seguem a mesma regra de atualização, possuem a mesma quantidade de vizinhos ou utilizam os mesmos parâmetros na regra de atualização da velocidade (exceto os parâmetros randômicos) Montes de Oca et al (2009b). O enxame como um todo se comporta da mesma forma, não existem especificidades entre as partículas.

Para que um enxame seja considerado heterogêneo ele deve conter no mínimo duas partículas diferentes, ou seja, o espaço de soluções deve ser analisado de forma diferente. Seja pela adoção de diferentes parâmetros de configuração, seja pelo modelo de vizinhança ou outro aspecto. Diante desta diferenciação foram sugeridas classificações quanto a heterogeneidade que um enxame pode apresentar, conforme Montes de Oca et al (2009b). Estas, por sua vez, são brevemente esplanadas a seguir.

5.1.1 Modelo de Influência

Neste tipo de heterogeneidade as partículas do enxame possuem diferentes mecanismos para definir como será a influência de uma partícula sobre as outras, ou seja, quanto uma partícula poderá interferir no movimento das demais. Para efeito de exemplo podemos citar o seguinte: em um enxame parte das partículas adota o modelo de topologia completamente conectada (também conhecido por *gbest*)

enquanto que as demais utilizam topologia anel (também chamada *lbest* ou *ring*) conforme mencionado na seção 2.2 PSO Padrão.

5.1.2 Regra de Atualização

Como o próprio nome sugere este tipo determina o uso de diferentes regras para a atualização da velocidade das partículas. Isto permite que o espaço de busca seja explorado de forma completamente diferente pelas partículas, obviamente que o desempenho desta abordagem dependerá da configuração adotada.

5.1.3 Heterogeneidade de vizinhança

Caracteriza-se pelos diferentes graus de vizinhança ao longo do enxame. Imagine que a população de partículas seja um grafo, para pertencer a esta classe de heterogeneidade ao menos duas partículas devem possuir diferentes graus de vizinhança.

5.1.4 Parâmetros

Configura-se por dispor de diferentes parâmetros de configuração para as partículas. Para que um enxame heterogêneo se enquadre nesta categoria é preciso que um subgrupo de partículas utilize a mesma regra de atualização e, no mínimo, duas delas utilizem diferentes parâmetros.

A heterogeneidade ainda pode sofrer uma nova classificação, quanto ao modo como as configurações são aplicadas ao longo do tempo (ciclo evolucionário). Por exemplo, para que tenhamos uma heterogeneidade *dinâmica*, as alterações devem ser realizadas à medida que as iterações avançam. Se isso não ocorre o tipo correspondente é o *estático*. Temos ainda o tipo *adaptativo*, podemos considerar este como um caso particular do tipo *dinâmico*, caracteriza-se por aplicar novas configurações em resposta a um determinado comportamento apresentado pelo enxame.

O restante do capítulo é organizado da seguinte forma: Seção 5.2 descreve os experimentos e é seguida da Seção 5.3 que apresenta os resultados, por fim a Seção 5.4 com a conclusão.

5.2 Experimentos

Nesta seção descrevemos como se deu o processo de treinamento das redes neurais MLP através da utilização de três algoritmos heterogêneos. Estes, por sua vez, foram compostos pelos algoritmos de otimização por enxames de partículas PSO padrão e FPSO descritos nas seções 2.2 e 2.3 respectivamente. O objetivo da aplicação dos enxames heterogêneos na tarefa de ajustar os pesos das conexões foi avaliar a capacidade de generalização da rede em relação ao algoritmo padrão.

Para representar as soluções nós utilizamos o mesmo modelo disposto na seção 3.2 e também as mesmas funções de custo, seção 3.3. A subseção 5.2.1 apresenta as configurações adotadas para a realização dos experimentos.

5.2.1 Configurações

Uma das ações necessárias para a construção de algoritmos heterogêneos é o estabelecimento da proporção entre os tipos. Para a realização dos experimentos foram utilizadas medidas similares as apresentadas em Montes de Oca et al (2009b). A proporção utilizada por cada algoritmo está indicada ao lado de cada tipo, por exemplo, $FPSO_{70}PSO_{30}$ significa que 70% do número total de partículas do enxame são do tipo FPSO e os 30% restantes são do tipo PSO padrão. A quantidade de partículas destinadas a cada um dos dois tipos, PSO padrão e FPSO, estão presentes na Tabela 16.

Para a realização dos experimentos nós adotamos grande parte das configurações utilizadas pelos algoritmos PSO padrão e FPSO quando estes foram aplicados separadamente no treinamento das redes neurais. Os parâmetros de configuração utilizados pelos algoritmos heterogêneos são apresentados na Tabela 15. Foram utilizadas as mesmas sete bases de dados descritas na Subseção 3.4.1 e a mesma disposição de arquitetura da rede, ou seja, redes com arquitetura fixa compostas por nº de características do problema (camada de entrada) – 6 neurônios (camada intermediária) – nº de classificações possíveis (camada de saída).

O critério de classificação aplicado aos algoritmos heterogêneos foi calculado conforme a regra do *winner-takes-all* – o mesmo critério adotado nos capítulos anteriores.

Tabela 15: Parâmetros de configuração dos algoritmos heterogêneos.

Algoritmo	Descrição	Valor
PSO	Critério de parada	100 iterações
	Medida de qualidade	NMSE
	Limite do espaço de busca	[-2.0, 2.0]
	Fatores de aceleração (c_1 e c_2)	$c_1 = c_2 = 1.4960$
	Peso de inércia (ω)	0,7298
FPSO	Critério de parada	100 iterações
	Medida de qualidade	NMSE
	Limite do espaço de busca	[-2.0, +2.0]
	Fator de inércia (ω)	[0.9 a 0.4]
	Soma dos coeficientes de aceleração (φ)	4.1
	K	Igual à quantidade de partículas do tipo FPSO
Rede Neural	Nº de neurônios escondidos	6
	Número máximo de iterações	100
	Funções de ativação	Tangente sigmóide – Linear
	Algoritmo de treino	Levenberg-Marquardt
	Nº de falhas de validação	5

Fonte: Autor (2011).

Tabela 16: Quantidade de partículas, por tipo, utilizadas pelos algoritmos heterogêneos no treinamento de redes neurais MLP.

Algoritmo	Nº de Partículas do tipo FPSO	Nº de Partículas do tipo PSO
$FPSO_{70}PSO_{30}$	21	9
$FPSO_{30}PSO_{70}$	9	21
$FPSO_{50}PSO_{50}$	15	15

Fonte: Autor (2011).

5.3 Resultados

Na Tabela 17 dispomos as médias e desvios-padrão do erro percentual de classificação dos algoritmos heterogêneos para as sete bases de dados descritas na Tabela 1. O número presente ao lado de cada variação da otimização por enxame de partículas indica a proporção seguida pelo algoritmo para distribuir as partículas, por exemplo, $FPSO_{70}$ indica que 70% das partículas são do tipo FPSO, ou seja, do total de 30 partículas, 21 são deste tipo.

Tabela 17: Média e desvio padrão do erro percentual de classificação para os três algoritmos heterogêneos, $FPSO_{70}$-PSO_{30}, $FPSO_{30}$-PSO_{70}, $FPSO_{50}$-PSO_{50}.

Base de Dados	$FPSO_{70}PSO_{30}$		$FPSO_{30}PSO_{70}$		$FPSO_{50}PSO_{50}$	
	Media	Desvio	Media	Desvio	Media	Desvio
Câncer	4,138	1,482	**3,755**	1,305	3,793	1,573
Diabetes	23,351	3,967	**22,500**	3,421	24,670	3,872
Coração	**21,290**	3,122	22,464	4,994	23,015	6,158
Vidros	**35,472**	8,516	42,076	9,912	40,189	9,597
Cavalos	35,202	6,495	33,919	5,486	**33,480**	5,534
Soja	38,686	6,341	**33,841**	4,433	39,506	6,836
Tireóide	5,785	1,996	5,002	2,238	**4,806**	1,874

Fonte: Autor (2011).

Tabela 18: Média e desvio padrão do tempo de execução, em segundos, para os três algoritmos heterogêneos, $FPSO_{70}$-PSO_{30}, $FPSO_{30}$-PSO_{70}, $FPSO_{50}$-PSO_{50}.

Base de Dados	$FPSO_{70}PSO_{30}$		$FPSO_{30}PSO_{70}$		$FPSO_{50}PSO_{50}$	
	μ	σ	μ	σ	μ	σ
Câncer	1288,996	203,601	650,219	87,511	641,984	54,402
Diabetes	796,080	25,318	852,351	53,702	1004,574	128,214
Coração	1213,322	41,176	1167,846	157,837	1228,124	84,366
Vidros	1010,646	51,518	965,613	79,517	1042,211	79,983
Cavalos	1635,130	46,599	1372,587	156,338	1487,576	100,453
Soja	18122,550	932,439	17632,490	2533,459	17068,465	875,474
Tireóide	6806,700	690,462	8475,499	906,362	9739,833	858,960

Fonte: Autor (2011).

Na Tabela 18 temos o tempo médio, em segundos, e desvio-padrão das execuções nas sete bases de dados. Na Tabela 19 dispomos os resultados dos testes

estatísticos, nela apenas os resultados relevantes estatisticamente foram relacionados, ou seja, apenas os resultados cujo algoritmo 1 foi melhor que o algoritmo 2.

Tabela 19: Resultado dos testes de hipótese para os algoritmos heterogêneos, $FPSO_{70}$-PSO_{30}, $FPSO_{30}$-PSO_{70}, $FPSO_{50}$-PSO_{50}, apenas os resultados no qual os algoritmos heterogêneos obtiveram melhor desempenho foram relacionados.

Base de dados	Algoritmo 1	Algoritmo 2	Valor t Calculado
Câncer	$FPSO_{30}PSO_{70}$	RN	-2,3582
Diabetes	$FPSO_{70}PSO_{30}$	RN	-2,9085
Vidros	$FPSO_{70}PSO_{30}$	$FPSO_{30}PSO_{70}$	-2,7680
	$FPSO_{70}PSO_{30}$	$FPSO_{50}PSO_{50}$	-2,0136
	$FPSO_{70}PSO_{30}$	LM	-3,6448
	$FPSO_{70}PSO_{30}$	Rprop	-4,8337
Cavalos	$FPSO_{50}PSO_{50}$	$FPSO_{Rprop}$	-2,1430
	$FPSO_{50}PSO_{50}$	LM	-6,9484
	$FPSO_{50}PSO_{50}$	Rprop	-2,6944
Soja	$FPSO_{30}PSO_{70}$	$FPSO_{70}PSO_{30}$	-3,4299
	$FPSO_{30}PSO_{70}$	$FPSO_{50}PSO_{50}$	-3,8083
	$FPSO_{30}PSO_{70}$	$FPSO_{Rprop}$	-19,0866
	$FPSO_{30}PSO_{70}$	$FPSO_{LM}$	-4,1661
	$FPSO_{30}PSO_{70}$	LM	-5,7043
	$FPSO_{30}PSO_{70}$	Rprop	-13,3457
	$FPSO_{30}PSO_{70}$	$FPSO:CG_{Lm}$	-4,8733
Tireóide	$FPSO_{50}PSO_{50}$	$FPSO_{Rprop}$	-3,3079
	$FPSO_{50}PSO_{50}$	Rprop	-4,0978

Fonte: Autor (2011).

Após a realização dos experimentos os algoritmos heterogêneos $FPSO_{70}PSO_{30}$, $FPSO_{30}PSO_{70}$ e $FPSO_{50}PSO_{50}$ foram comparados com os algoritmos relacionados na Seção 3.5. Os resultados estão listados na Tabela 20.

De acordo com a Tabela 20, é possível verificar que os algoritmos heterogêneos alcançaram as menores médias de erros em três das sete bases de dados. Cada proporção do algoritmo heterogêneo foi melhor em uma determinada base de dados, não tendo sido possível identificar uma proporção ideal para todas as bases testadas.

Tabela 20: Média e desvio padrão nos algoritmos heterogêneos em relação aos algoritmos $FPSO_{LM}$, $FPSO_{RPROP}$, LM, RPROP e $FPSO:CG_{Lm}$.

		Câncer	Diabetes	Coração	Vidros	Cavalos	Soja	Tireóide
$FPSO_{LM}$	μ	**2,969**	22,708	**18,580**	35,598	34,762	38,157	**1,774**
	s	1,688	2,932	3,319	9,160	5,174	3,542	0,398
$FPSO_{RPROP}$	μ	3,352	22,778	21,058	39,748	36,557	63,157	5,978
	s	1,484	3,595	6,023	10,821	5,588	7,150	0,504
LM	μ	4,705	26,163	21,986	43,270	43,150	40,510	3,556
	s	1,780	3,508	3,237	8,050	5,242	4,621	1,203
RPROP	μ	4,076	24,080	19,710	47,044	37,106	61,706	6,337
	s	1,774	3,057	2,985	9,971	4,869	10,542	0,822
$FPSO:CG_{Lm}$	μ	3,886	**21,979**	19,159	36,855	35,018	41,784	4,526
	s	1,337	4,711	3,189	9,133	5,813	7,749	1,923
$FPSO_{70}PSO_{30}$	μ	4,138	23,351	21,290	**35,472**	35,202	38,686	5,785
	s	1,482	3,967	3,122	8,516	6,495	6,341	1,996
$FPSO_{30}PSO_{70}$	μ	3,755	22,500	22,464	42,076	33,919	**33,841**	5,002
	s	1,305	3,421	4,994	9,912	5,486	4,433	2,238
$FPSO_{50}PSO_{50}$	μ	3,793	24,670	23,015	40,189	**33,480**	39,506	4,806
	s	1,573	3,872	6,158	9,597	5,534	6,836	1,874

Fonte: Autor (2011).

O teste de hipótese foi aplicado para verificar o quão bom foi o desempenho dos algoritmos heterogêneos em relação aos demais algoritmos relacionados na Tabela 20. Apenas o resultado no qual o algoritmo heterogêneo foi comprovadamente melhor aos demais está relacionado na Tabela 21.

Tabela 21: Resultado do teste de hipótese no qual o algoritmo heterogêneo obteve melhor desempenho em relação aos demais testados neste trabalho para o treinamento de redes neurais MLP.

Base de dados	Algoritmo 1	Algoritmo 2	Valor t Calculado
Soja	$FPSO_{30}PSO_{70}$	$FPSO_{LM}$	-4,166

Fonte: Autor (2011).

Conforme a Tabela 21 o algoritmo $FPSO_{30}PSO_{70}$ foi o único algoritmo heterogêneo com desempenho comprovadamente melhor em relação aos demais algoritmos testados no Capítulo 3, nos demais casos os algoritmos heterogêneos apresentaram equivalência estatística e alguns casos, produziram resultados piores.

5.4 Conclusão

Neste capítulo fizemos uso de uma abordagem apresentada em Montes de Oca et al (2009b). Este trabalho objetiva aplicar especificidade ao nível de partícula, promovendo assim algumas diferenciações. A intenção é possuir ao menos duas partículas que analisam de forma diferenciada o espaço de soluções. Diferentes parâmetros podem ser adotados para que um enxame possa ser considerado heterogêneo. Por exemplo, podemos ter duas ou mais partículas que possuem diferentes regras de atualização ou mesmo possuam diferentes parâmetros de configuração em relação as demais partículas do enxame.

Nos experimentos que utilizaram enxame de partículas heterogêneo foram criados três novos algoritmos. Estes algoritmos caracterizam-se por possuírem diferentes concentrações de tipos, são eles: $FPSO_{70}PSO_{30}$, $FPSO_{30}PSO_{70}$ e $FPSO_{50}PSO_{50}$.

Os resultados dos testes de hipótese realizados nos algoritmos $FPSO_{70}PSO_{30}$, $FPSO_{30}PSO_{70}$ e $FPSO_{50}PSO_{50}$ comprovaram que o uso de enxames heterogêneos é capaz de melhorar a capacidade de generalização de uma rede neural (algoritmo $FPSO_{30}PSO_{70}$ aplicado à base de dados Soja), no entanto houve casos em que a utilização da técnica produziu resultados piores.

No decorrer destes experimentos não foi possível identificar uma proporção ideal entre os dois tipos de PSO, dentre as três utilizadas (30%, 50% e 70%), capaz de produzir resultados melhores, estatisticamente, em todas as bases de dados. Este problema, com certeza, será um dos objetos de estudo para os enxames heterogêneos.

Capítulo 6

Conclusões e Trabalhos Futuros

Este capítulo aponta os resultados obtidos durante esta pesquisa; identifica pontos de melhorias e também novas oportunidades de estudo levantadas durante os experimentos.

6.1 Conclusões

Esta dissertação abordou uma tarefa importante na área de aprendizado supervisionado, treinamento de redes neurais MLP e a definição da arquitetura e ajuste dos pesos sinápticos, para problemas de classificação de padrões (BRAGA; CARVALHO; LUDERMIR, 2007).

Para tanto fizemos uso de duas diferentes abordagens durante o treinamento das redes neurais MLP. Foram utilizados enxames não-heterogêneos e enxames heterogêneos. Para os algoritmos não-heterogêneos foram utilizados exames do tipo Frankenstein PSO – FPSO (MONTES DE OCA et al, 2009a) e algumas variações propostas; $FPSO_{Lm}$ – Frankenstein PSO associado ao *Levenberg-Marquardt*, $FPSO_{Rprop}$ – Frankenstein PSO associado ao *Resilient-Backpropagation* e $FPSO{:}CG_{Lm}$ – Frankenstein PSO com convergência garantida associado ao *Levenberg-Marquardt*). Também foram utilizados os algoritmos LM (*Levenberg-Marquardt*) e Rprop (*Resilient back-propagation*). Em um segundo momento foram utilizados os enxames heterogêneos (MONTES DE OCA et al, 2009b) – utilizamos esta nomenclatura quando em um enxame ao menos duas partículas analisam o espaço de buscas de maneiras diferentes. Enxames heterogêneos podem ser classificados de diferentes maneiras, como por exemplo: quanto ao modelo de influência, a regra de atualização, a heterogeneidade de vizinhança e aos parâmetros de configuração.

Os exames heterogêneos foram definidos seguindo a seguinte proporção: $FPSO_{70}PSO_{30,}$ $FPSO_{30}PSO_{70}$ e $FPSO_{50}PSO_{50}$. O número ao lado do FPSO ou PSO

identifica a porcentagem utilizada para cada tipo, por exemplo: $FPSO_{30}PSO_{70}$ indica que 30% do total de partículas do enxame são do tipo FPSO e o restante, 70%, são do tipo PSO.

Os algoritmos desenvolvidos aqui para ajuste automático das arquiteturas e ajuste dos pesos das conexões da rede neural - PSO-$FPSO_{Lm}$ e FPSO-$FPSO_{Lm}$ - basearam-se na metodologia disposta em Carvalho (2007). Na qual dois algoritmos PSO são utilizados simultaneamente para definir a arquitetura e treinar as redes neurais MLP (nesta fase para compor nosso enxame utilizamos apenas partículas do tipo $FPSO_{Lm}$, que foram aquelas que obtiveram o melhor desempenho na fase de treinamento das redes).

Este processo de treinamento se deu em duas fases: na primeira foram avaliados os desempenhos de algumas variações do PSO e dois algoritmos de busca local no ajuste dos pesos da rede neural com arquitetura fixa. Em um segundo momento foram utilizados os algoritmos PSO-$FPSO_{Lm}$ e FPSO-$FPSO_{Lm}$ para definir o número de neurônios na única camada intermediária considerada e treinar as redes neurais.

Os algoritmos utilizados na primeira parte foram: $FPSO_{Lm}$, $FPSO_{Rprop}$, $FPSO{:}CG_{Lm}$, *Resilient back-propagation* (Rprop) e *Levenberg-Marquardt* (LM). O uso do Frankenstein PSO proporcionou melhores resultados porque para cada fase do processo evolucionário acentuou-se a influência de determinado comportamento na composição da solução (o fato de o FPSO apresentar inicialmente uma topologia completamente conectada favoreceu uma rápida propagação da melhor solução. Esta configuração aplicada nas iterações iniciais propiciou ao algoritmo a chance de encontrar soluções de boa qualidade. Em contrapartida a topologia anel retardou a propagação da melhor região encontrada nas iterações finais – propiciando maior explotação - associada a isto o uso do peso de inércia baixo evitou que o restante do enxame se locomovesse a regiões menos promissoras).

Realizamos avaliação experimental e dois critérios foram usados para medir o desempenho dos algoritmos: erro percentual de classificação e o tempo de execução. Essa avaliação permitiu concluir que o algoritmo FPSO-$FPSO_{Lm}$ obteve melhor

acurácia de classificação se comparado ao algoritmo PSO-FPSO$_{Lm}$ em contrapartida este último apresentou menor tempo de execução.

A utilização de enxames heterogêneos na fase de treinamento da rede neural proporcionou boas médias de erros de classificação, no entanto não foi possível estabelecer uma proporção ideal aplicável a todas as bases de dados testadas.

6.2 Trabalhos Futuros

Os bons resultados obtidos pelos algoritmos propostos incentivam o aprimoramento das pesquisas sobre o assunto, principalmente no estudo de meios que possam melhorar o tempo de execução. Uma solução a este problema poderia ser a reimplementação dos algoritmos em linguagens de baixo nível.

Outra ideia é investir no estudo de uma redução no tamanho do enxame de partículas ao longo das iterações, mantendo somente as ‘x’ melhores partículas (podemos considerar a definição deste ‘x’ como uma nova linha de pesquisa), o que garantiria mais iterações de avaliação usando o mesmo tempo de execução atingido hoje (isto porque o tempo investido nas partículas que não produzem bons resultados seria revertido para aquelas localizadas em regiões mais promissoras do espaço de busca).

Avaliar a aplicação dos algoritmos heterogêneos e não-heterogêneos ou, neste caso, homogêneos em outras classes de problemas; otimizar outros parâmetros como taxa de aprendizado, quantidade de camadas intermediárias, funções de ativação, algoritmos de treinamento; bem como propor novas combinações para os algoritmos heterogêneos. A exemplo podemos ter combinações entre algoritmos FPSO e PSO *Barebones* (O PSO *Barebones* substitui as equações de atualização da velocidade e posição por um método estatístico), o que geraria o “FPSO-PSO$_{Barebones}$” ou mesmo PSO com outras técnicas de otimização.

Outra linha de pesquisa pode estar voltada ao estudo das concentrações por tipo em algoritmos heterogêneos. Evitando-se a experimentação em busca de uma quantidade ideal para determinado tipo de problema. Uma solução seria compor um algoritmo capaz de controlar dinamicamente as concentrações dos tipos, análogo ao

que é feito hoje pela convergência garantida em que uma variável controla o tamanho do raio de busca de uma partícula.

O próximo passo como continuidade a este trabalho é a aplicação dos algoritmos heterogêneos na definição das arquiteturas da rede neural MLP. Verificar seu comportamento nos dois contextos, ajuste dos pesos sinápticos e definição do número de camadas e neurônios escondidos.

REFERÊNCIAS

ALMEIDA, L. M.; LUDERMIR, T. B., A hybrid method for searching near-optimal artificial neural networks. In: **International Conference on Hybrid Intelligent Systems**, 6., 2006, Rio de Janeiro. [S.l.: s.n] p 36-39.

ALMEIDA, L. M.; LUDERMIR, T. B., A Multi-Objective Memetic and Hybrid Methodology for Optimizing the Parameters and Performance of Artificial Neural Networks. **Neurocomputing Journal**. Disponível em: <www.elsevier.com/locate/neucom>. Acesso: 09 jan. 2010. p 1438-1450.

ALMEIDA, L. M., **Uma Metodologia de Busca por Redes Neurais Artificiais Quase-Ótimas**. 2011. 102 f. Dissertação (Mestrado em Ciência da Computação)- Centro de Informática, Universidade Federal de Pernambuco, Recife, 2011.

ATYABI, A.; PHON-AMNUAISUK, S.; HO, C. K., Cooperative Learning of Homogeneous and Heterogeneous Particles in Area Extension PSO. In: **IEEE Congress on Evolutionary Computation (CEC)**, 2008, Hong Kong. [S.l.: s.n] p 3889-3896.

BERTSIMAS, D.; TSITSIKLIS, J., Simulated Annealing. In: **Statistical Science**. [S.l.:s.n], 1993. p 10-15. v. 8, n.1.

BRAGA, A. P.; CARVALHO, A. P. L. F.; LUDERMIR, T. B., **Redes Neurais Artificiais – Teoria e Aplicações**. [S.l.]: LTC, 2007. 248 p. 2 ed.

CARVALHO, M.; LUDERMIR, T. B., Hybrid Training of Feed-Forward Neural Networks with PSO. In: **International Conference on Neural Information Processing (ICONIP)**. [S.l.:s.n], 2006. v. 4233, p 1061-1070.

CARVALHO, M.; LUDERMIR, T. B., An Analysis Of PSO Hybrid Algorithms For Feed-Forward Neural Networks Training. In: **Brazilian Symposium on Neural Networks (SBRN)**, 9., Ribeirão Preto: [s.n], 2006. p 6-11.

CARVALHO, M.; LUDERMIR, T. B., Particle Swarm Optimization of Feed-Forward Neural Networks with Weight Decay. In: **International Conference on Hybrid Intelligent Systems (HIS)**. Rio de Janeiro: [s.n], 2006. p 5-8.

CARVALHO, M. R., **Uma Análise da Otimização de Redes Neurais MLP por Enxame de Partículas**. 2007. 66 f. Dissertação (Mestrado em Ciência da Computação)-Centro de Informática, Universidade Federal de Pernambuco, Recife, 2007.

CARVALHO, M.; LUDERMIR, T. B., Particle Swarm Optimization of Neural Network Architectures and Weights. In: **International Conference on Hybrid Intelligent Systems (HIS)**. Washington: [s.n], 2007. p 336-339.

CHAURASIA, S.; DAWARE, S., Implementation of Neural Network in Particle Swarm Optimization (PSO) Techniques. In: **International Conference on Intelligent Agent & Multi-Agent Systems (IAMA)**. Chennai: [s.n], 2009. p 1-2.

CLERC, M.; KENNEDY, J., The particle swarm–explosion, stability, and convergence in a multidimensional complex space. **IEEE Transactions on Evolutionary Computation Journal**, New Jersey, Feb. 2002. Disponível em: <http://dl.acm.org/citation.cfm?id=2221574>. Acesso em: 19 maio 2010. . v. 6, n. 1. p 58-73.

Eberhart, R.; Shi, Y., Comparing Inertia Weights and Constriction Factors in Particle Swarm Optimization. In: **Proceedings of the Congress on Evolutionary Computation**, 2000, La Jolla, CA [s.n]. v. 1, p 84-88.

EBERHART, R.; SHI, Y., Tracking and optimizing dynamic systems with particle swarms, In: **Proceedings of the Congress on Evolutionary Computation**, 2001, Seoul. [S.l.:s.n] v. 1, p 94-100.

ENGELBRECHT, A., Scalability of A Heterogeneous Particle Swarm Optimizer. In: **IEEE Symposium on Swarm Intelligence (SIS)**, Paris: [s.n], 2011. p 1-8.

A. FRANK; A. ASUNCION., **UCI Machine Learning Repository**, California University, Sch. Inform. Comput. Sci., Irvine, CA: [s.n], 2007. Disponível em: <http://archive.ics.uci.edu/ml>. Acesso em: 01 jun. 2010.

GIMMLER, J.; STÜTZLE, T.; EXNER, T. E., Hybrid Particle Swarm Optimization: An examination of the influence of iterative improvement algorithms on Performance. In: **International Workshop (ANTS)**, 6., [S.l.:s.n], 2006. v. 4150 p 436-443.

GLOVER, F.; LAGUNA, M., **Tabu Search**. Massachusetts: Kluwer Academic Publishers Norwell, 1997. 382 p.

GOMES, G. S. S.; LUDERMIR, T. B., Aproximação de funções através de redes neurais artificiais com funções de ativação complemento log-log e probit. In: Workshop on Computational Intelligence (WCI), 2., 2008, Salvador: [s.n]. **Anais...** Porto Alegre: Sociedade Brasileira de Computação, 2008.

GUDISE, V. G.; VENAYAGAMOORTHY, G. K., Comparison of Particle Swarm Optimization and backpropagation as training algorithms for neural networks. In: **Swarm Intelligence Symposium (SIS)**, 3., Indianapolis: [s.n], 2003. p 100-117.

HAYKIN, S., **Neural Networks: A Comprehensive Foundation**. [S.l.:] Prentice Hall, 1999, 842 p. Second edition.

KARABOGA, D.; BASTURK, B., On the performance of artificial bee colony (ABC) algorithm. In: **Applied Soft Computing**, [S.l.:s.n], 2008. p 687-697. v. 8, issue 1.

KENNEDY, J.; EBERHART, R., Particle Swarm Optimization. In: **IEEE International Conference on Neural Networks**, Piscataway: [s.n], 1995. p 1942-1948.

KIRANYAZ, S.; INCE, T.; YILDIRIM, A.; GABBOUJ, M., Evolutionary artificial neural networks by multi-dimensional particle swarm optimization. In: **Neural Networks Journal**, [S.l.:s.n], 2009. v. 22, Issue 10, Disponível em: <http://www.sciencedirect.com/science/article/pii/S0893608009001038> v. 22 n. 10, p 1448-1462. Acesso em: 14 dez 2010.

KIRKPATRICK, S.; GELLAT JR, C. D.; VECCHI, M. P., Optimization by Simulated Annealing. In: **Science**, 1983, [S.l.:] New Series. v. 220, n. 4598. p 671-680.

LIMA, N. F.; LUDERMIR, T. B., Frankenstein PSO Applied to Neural Network Weights and Architectures. In: **IEEE Congress on Evolutionary Computation (CEC)**, 2011, New Orleans: [s.n], 2011. p 2452-2456.

LING, S. H.; NGUYEN, H. T.; CHAN, K., A New Particle Swarm Optimization Algorithm for Neural Network Optimization. In: **International Conference on Network and System Security**, 3., 2009, [S.l.:s.n], 2009. v. 3971, Issue 5-6, p 516 - 521.

LUDERMIR, T. B.; YAMAZAKI, A.; ZANCHETTIN, C., An Optimization Methodology for Neural Network Weights and Architectures. In: **IEEE Transactions on Neural Networks**, 2006, [S.l.:s.n], 2006. v. 17 n. 6, p 1452-1459.

MENDES, R.; KENNEDY, J.; NEVES, J., The Fully Informed Particle Swarm: Simpler, Maybe Better. In: **IEEE Transactions Evolucionary Computation**, 2004, [S.l.:s.n], 2004. v. 8 n. 3, p. 204-210.

MIDDENDORF, M.; JANSON, S., A Hierarchical Particle Swarm Optimizer and Its Adaptive Variant. In: **IEEE Transactions on Systems, Man, and Cybernetics. Part B – Cybernetics**, 2005, [S.l.:s.n], 2005. v. 35 n. 6, p. 1272-1282.

MONTES DE OCA, M.; STÜTZLE, T.; BIRATTARI, M.; DORIGO, M., Frankenstein's PSO: A Composite Particle Swarm Optimization Algorithm. In: **IEEE Transactions on Evolutionary Computation**, 2009, [S.l.:s.n], 2009. v 13 n. 5, p 1120-1132.

MONTES DE OCA, M. A.; PEÑA, J.; STÜTZLE, T.; PINCIROLI, C.; DORIGO, M., Heterogeneous Particle Swarm Optimizers. In: **IEEE Congress on Evolutionary Computation (CEC)**, 2009, Trondheim: [s.n], 2009. p 698-705.

NINOMIYA, H.; ZHANG, Q.-J., Particle with Abitlity of Local search Swarm Optimization: PALSO for Training of *Feed-forward* Neural Networks. In: **International Joint Conference on Neural Networks (IJCNN)**, 2008, Hong Kong: [s.n], 2008. p 3009-3014.

PEER, E.; BERGH, F. D.; ENGELBRECHT, A., Using Neighbourhoods with the Guaranteed Convergence PSO. In: **Proceedings of the Swarm Intelligence Symposium (SIS)**, 2003, [S.l.:s.n], 2003. p 235-242.

PINGZHOU T.; ZHAOCAI X., The Research on BP Neural Network Model Based on Guaranteed Convergence Particle Swarm Optimization. In: **International Symposium on Intelligent Information Technology Application**, 2., 2008, Shanghai: [s.n], 2008. v. 2. p 13-16.

PRECHEIT, L., Proben1 – A Set of Neural Network Benchmark Problems and Benchmark Rules. In: **Fakultät für infortamik**, 1994, Universität Karlsruhe: [s.n], 1994. 38 p.

PRUDÊNCIO, R. C., **Projeto Híbrido de Redes Neurais**. 2002. 101 f. Dissertação (Mestrado em Ciência da Computação)-Centro de Informática, Universidade Federal de Pernambuco, Recife, 2002.

RATNAWEERA, A.; HALGAMUGE, S. K.; WATSON, H. C., Self-organizing hierarchical particle swarm optimizer with time-varying acceleration coefficients. In: **IEEE Transactions Evolution Computation**, 2004, [S.l.:s.n], 2004. v. 8 n. 3. p. 240-255.

SHI, Y.; EBERHART, R., A Modified Particle Swarm Optimizer. In: **IEEE World Congress on Computational Intelligence**, 1998, Anchorage: [s.n], 1998. p 69-73.

SHI, Y.; EBERHART, R., Empirical Study of Particle Swarm Optimization. In: **IEEE Congress Evolutionary Computation**, 1999, Washington: [s.n], 1999. v. 3 p 1945-1950.

SILVA, A. J., MINEU, N. L., LUDERMIR, T. B., Evolving Artificial Neural Networks Using Adaptive Differential Evolution. In: **Proceedings of Ibero-American conference on Advances in artificial intelligence (IBERAMIA)** 12., Bahía Blanca, Argentina: [s.n], 2010. v. 6433. p 396-405.

Sivagaminathan, R. K.; Ramakrishnan, S., A hybrid Approach for Feature Subset Selection Using Neural Networks and Ant Colony Optimization. In: **Expert Systems with Applications**, 2007, [S.l.:s.n], 2007. v. 33 n. 1. p 49-60.

STORN, R., Differential evolution - A Simple and Efficient Adaptive Scheme for Global Optimization Over Continuous Spaces. In: **International Computer Science Institute**, 1995, Berkeley: [s.n], 1995. Technical Report TR-95-012, ICSI. 15 p. Disponivel em: <ftp.icsi.berkeley.edu>. Acesso em: 22 fev 2011.

TEIXEIRA, L.; OLIVEIRA, F.; OLIVEIRA, A.; BASTOS FILHO, C., Adjusting Weights and Architecture of Neural Networks through PSO with Time-Varying Parameters and Early Stopping. In: **Brazilian Symposium on Neural Networks (SBRN**), 10., 2008, Salvador: [s.n], 2008. p 33-38.

TEODOROVIC, D.; LUCIC, P.; MARKOVIC, G.; DELL' ORCO, M., Bee Colony Optimization: Principles and Applications. In: **Seminar on Neural Network Application in Eletrical Engineering (NEUREL)** 8., 2006, Belgrade, Serbia & Montenegro: [s.n], 2006. p 151-156.

VALENÇA, I. C., **Modelos Híbridos Baseados em Redes Neurais, Lógica Fuzzy e Busca para Previsão de Séries Temporais.** 2010. 83 f. Dissertação (Mestrado em Ciência da Computação)-Centro de Informática, Universidade Federal de Pernambuco, Recife, 2010.

VALENÇA, I. C.; LUDERMIR, T. B.; VALENCA, M., Hybrid Systems to Select Variables for Time Series Forecasting Using MLP and Search Algorithms. In: **Brazilian Symposium on Neural Networks (SBRN)**, 11., 2010, São Paulo: [s.n], 2010. p 247-252.

VAN WYK, A. B.; ENGELBRECHT, A. P., Overfitting by PSO Trained Feed-forward Neural Networks. In: **IEEE Congress on Evolutionary Computation (CEC)**, 2010, Barcelona: [s.n], 2010. p 1-8.

WAYNE, D. W., **Applied Nonparametric Statistics**. [S.l.:] Duxbury Resource Center, 1990. 656 p. Second edition.

YAMAZAKI, A., **Uma Metodologia para Otimização de Arquiteturas e Pesos de Redes Neurais**. 2004. 135 f. Tese (Doutorado em Ciência da Computação)-Centro de Informática, Universidade Federal de Pernambuco, Recife, 2004.

YANAN, M.; XIUWEI, F.; LI, F., Application of Neural Network Trained by Adaptive Particle Swarm Optimization to Fault Diagnosis for Steer-by-Wire System. In: **International Conference on Measuring Technology and Mechatronics Automation, (ICMTMA)**, 2010, Changsha: [s.n], 2010. v. 1. p 652-655.

YAO, X.; LIU, Y., A New Evolutionary System for Evolving Artificial Neural Networks. In: **IEEE Transactions on Neural Networks**, 1997, [S.l.:s.n], 1997. v. 8 n. 3. p 694-713.

ZANCHETTIN, C.; LUDERMIR, T. B., Técnica Híbrida de Otimização para Pesos e Arquiteturas de Redes Neurais Artificiais. In: **Congresso da Sociedade Brasileira de Computação**, 25, 2005, Rio Grande do Sul: [s.n], 2005. p 902-911.

ZANCHETTIN, C.; LUDERMIR, T. B., Hybrid Optimization Technique for Artificial Neural Networks. In: **International Conference on Enterprise Information Systems**, 2009, Milão: [s.n], 2009. p 242-247.

ZANCHETTIN, C.; LUDERMIR, T. B.; ALMEIDA, L. M., Hybrid Training Method for MLP: Optimization of Architecture and Training. In: **IEEE Trasactions on Systems,**

Man, and Cybernetics - Part B: Cybernetics, 2011, [S.l.:s.n], 2011. v. 41 n. 4. p 1097-1109.

ZARTH, A. M.; LUDERMIR, T. B., Optimization of Neural Networks Weights and Architecture: A Multimodal Methodology. In: **International Conference on Intelligent Systems Design and Applications (ISDA)**, 9., 2009, Pisa, Italy: [s.n], 2009. p 209-214.

ZARTH, A. M., **Otimização Evolucionária Multimodal de Redes Neurais Artificiais com Evolução Diferencial**. 2010. 71 f. Dissertação (Mestrado em Ciência da Computação)-Centro de Informática, Universidade Federal de Pernambuco, Recife, 2010.

ZHANG, W.; LIU, Y.; CLERC, M., An Adaptive PSO Algorithm for Reactive Power Optimization. In: **International Conference on Advances in Power System Control, Operation and Management (APSCOM**), 6., 2003, Hong Kong: [s.n], 2003. p 302-307.

ZHENG, Y.-L; MA, L.-H; ZHANG, L.-Y; QIAN, J.-X., Empirical study of particle swarm optimizer with an increasing inertia weight. In: **IEEE Congress on Evolutionary Computation (CEC)**, 2003, [S.l.:s.n], 2003. v. 1. p 221–226.

ZHONG, B.; WANG, D.; LI, T., Application of Optimized Neural Network Based on Particle Swarm Optimization Algorithm in Fault Diagnosis. In: **IEEE International Conference on Cognitive Informatics (ICCI)**, 8., 2009, Kowloon: [s.n], 2009. p 476-480.