X Congresso Brasileiro de Inteligência Computacional (CBIC'2011), 8 a 11 de Novembro, 2011, Fortaleza, Ceará, Brasil

Aplicação de Otimização por Enxame de Partículas para Treinamento de Redes Neurais MLPs

Saulo M. O. C. dos Santos, Mêuser J. Valença e Carmelo J. A. Bastos-Filho

Escola Politécnica de Pernambuco – Universidade de Pernambuco (UPE)
CEP: 50720-001 – Recife – PE – Brasil
{smocs, meuser, carmelofilho}@ecomp.poli.br

Resumo – Redes Neurais Artificiais têm sido amplamente utilizadas para resolver problemas de classificação e previsão. Muitas aplicações desta técnica apresentaram resultados mais precisos quando comparados aos resultados obtidos pelas técnicas estatísticas tradicionais. Para que a sua utilização seja generalizada, ou seja, para que seja possível resolver problemas não linearmente separáveis é preciso usar o *Multilayer Perceptron*. O algoritmo mais comum para treinar este tipo de rede neural é conhecido como *Backpropagation*. Entretanto, este algoritmo apresenta alguns inconvenientes, como convergência lenta e a possibilidade de ficar preso facilmente em mínimos locais. Este trabalho propõe aplicar algumas variações adaptativas recentemente propostas do algoritmo original de otimização por enxames de partículas para treinar redes neurais. O algoritmo ClanAPSO mostrou-se eficaz na tarefa de treinamento da rede neural, apesar de seu tempo e quantidade de iterações para convergência terem aumentado significativamente.

Palavras-chave – Redes neurais artificiais, Perceptron de múltiplas camadas, Otimização por enxames de partículas.

Abstract –Artificial Neural Networks have been widely adopted to tackle problems of classification and prediction. Many applications of this technique presented more accurate results when compared to traditional statistical techniques. In order to solve general problems, one needs to use the Multilayer Perceptron. The most common algorithm to train this type of neural network is called Backpropagation. However, this algorithm presents some drawbacks, such as slow convergence and the possibility to get stuck in local minima. This paper proposes to apply some recently proposed adaptive variations of particle swarm based algorithms to train neural networks. The ClanAPSO algorithm presented better results in the task of training the neural network, although its duration and number of iterations until convergence increased significantly.

Keywords – Artificial Neural Networks, Multi-Layer Perceptron, Particle Swarm Optimization.

1 Introdução

Redes Neurais Artificiais (RNAs) são sistemas paralelos distribuídos compostos por unidades de processamento simples (nodos) que calculam determinadas funções matemáticas (normalmente não lineares). Tais unidades são dispostas em uma ou mais camadas e interligadas por um grande número de conexões, geralmente unidirecionais. Na maioria dos modelos essas conexões são associadas a pesos, os quais armazenam o conhecimento representado no modelo e servem para ponderar a entrada recebida por cada neurônio da rede. O funcionamento destas redes é inspirado em uma estrutura física concebida pela natureza: o cérebro humano [1].

No processo de execução das RNAs, existe a fase de aprendizagem, na qual um conjunto de exemplos de entradas com as respectivas saídas é apresentado à rede neural. Após o cálculo da saída para cada exemplo, é realizada uma comparação com seus respectivos resultados, que por sua vez já são conhecidos. Esta operação permite que a rede ajuste seus pesos para conseguir um resultado mais próximo do desejado.

O algoritmo *Backpropagation* é comumente utilizado para realizar este processo de aprendizado. Este algoritmo consiste na generalização da regra delta, ou também conhecido como técnica do gradiente descendente. Sua execução se dá através de duas etapas: na primeira etapa, calcula-se o sinal de saída e o erro propagando o sinal da entrada para a saída; na segunda etapa os erros são propagados da saída para a entrada, ajustando os valores dos pesos de acordo com a regra delta generalizada [2].

Com a vasta utilização do *Backpropagation*, foram encontrados alguns pontos negativos no seu desempenho como: convergência lenta e possibilidade de ficar preso facilmente em mínimos locais [3], especialmente em problemas de alta dimensionalidade, isto é, quando a arquitetura da rede neural possui uma grande quantidade de pesos e elevado número de exemplos para treinamento. O algoritmo *Backpropagation* também não apresenta bom desempenho para problemas com múltiplos objetivos conflitantes.

Em função das limitações do algoritmo *Backpropagation*, várias técnicas de otimização para realizar o treinamento de redes neurais têm sido propostas. Dentre estas, pode-se citar: Otimização por Enxames de Partículas [3]; a técnica do Time Assíncrono [4], que combina várias técnicas; Otimização por Colméias [5], entre outras.

Este trabalho visa comparar o desempenho do treinamento de Redes Neurais Artificiais *Multilayer Perceptron* (MLP) por diferentes algoritmos de Otimização por Enxame de Partículas (PSO), em especial o ClanPSO e o ClanAPSO, e verificar a viabilidade de seu uso para treinamento de uma rede neural como uma alternativa ao algoritmo *Backpropagation*.

Este artigo está organizado na seguinte forma: na seção 2 e 3 são apresentados os conceitos sobre redes neurais artificiais; nas seções 4, 5, 6, 7 e 8 são apresentados o PSO, o APSO, as topologias de comunicação usadas pelo PSO, o ClanPSO e o ClanAPSO, respectivamente. Na seção 9 são apresentadas as configurações usadas nas simulações realizadas. Na seção 10 são apresentados os resultados das simulações. E finalmente, na seção 11 são apresentados as conclusões e ideias para trabalhos futuros.

2 *Multilayer Perceptron* (MLP)

O modelo de RNA *Perceptron*, é uma rede simples onde existem várias unidades de processamento conectadas a uma camada de saída. Uma MLP é uma generalização da rede *Perceptron* com a adição de pelos menos uma camada intermediária. Esta camada intermediária é a responsável pela representação de não linearidade na rede, o que permite que as redes MLP possam resolver problemas não linearmente separáveis.

Em uma rede multicamadas, o processamento realizado por cada neurônio é definido pela combinação dos processamentos realizados pelos neurônios da camada anterior que estão conectadas ao mesmo [1].

Para que uma rede neural consiga bons resultados, é necessário que os pesos de suas conexões sejam ajustados até que se estabeleça um conjunto de pesos ótimos em função da minimização de uma função objetivo. Este processo de minimização para estabelecimento dos pesos ótimos é chamado de treinamento da rede neural, e é realizado através da execução de um algoritmo de otimização. O algoritmo tradicionalmente utilizado para treinamento das redes MLP consiste numa generalização da regra delta, conhecido como algoritmo *Backpropagation* [2].

3 Backpropagation

As redes MLP são redes com treinamento supervisionado, de tal forma que durante o treinamento destas redes com pelo menos uma camada intermediaria não se conhece o erro desta camada, sendo este conhecimento necessário para realizar o reajuste exato dos pesos. O algoritmo *Backpropagation* aborda este problema realizando uma propagação recursiva dos erros [2].

A aprendizagem da rede ocorre em duas etapas: a etapa *forward* e a etapa *backward*. Na primeira etapa, os sinais são propagados no sentido progressivo para definir a saída da rede. Ao final desta etapa, pode-se calcular o erro na camada de saída pela diferença entre a saída desejada e a calculada. Na etapa seguinte, o erro é propagado recursivamente da saída até a entrada permitindo assim que seja realizado o ajuste dos pesos através da regra delta generalizada.

Um ponto importante a ser considerado durante o treinamento de uma rede neural é que o aprendizado demasiado da rede pode causar *overfitting*, ou seja, após certa quantidade de ciclos de aprendizado, a rede pode começar a memorizar os exemplos de entrada. Isto tem como consequência uma pior capacidade de generalização da rede. Para evitar isto, durante o treinamento de uma MLP se utiliza um critério de parada. Um critério de parada bastante popular é conhecido como validação cruzada.

4 Algoritmo PSO

Otimização por enxames de partículas (PSO) é uma técnica de inteligência computacional proposta por Kennedy e Eberhart em 1995 [7]. Esta técnica é comumente utilizada para resolver problemas de otimização em espaços contínuos e hiperdimensionais. Ela foi inspirada no comportamento de bandos de aves. A ideia básica do PSO é criar partículas que simulem os movimentos dos pássaros para alcançar algo dentro de um espaço de busca específico. A técnica explora o comportamento social dentro de um grupo organizado de indivíduos e sua capacidade de comunicação. Cada partícula modifica sua velocidade levando em consideração a melhor posição da partícula e a melhor posição do grupo ao longo do processo de busca.

Um enxame é formado por uma população de partículas, onde cada partícula é composta de: um vetor de posição $\vec{x}\ (t)$ que representa uma possível solução para um problema; o valor obtido a partir da valoração da função objetivo nesta posição$f(\vec{x}\ (t))$; uma memória cognitiva que armazena a melhor posição já visitada pela partícula ao longo das iterações do algoritmo, e o valor da função objetivo para essa posição ($\vec{x}_{pbest}$; $f(\vec{x}_{pbest})$); uma vizinhança, que é entendida como o conjunto de partículas do enxame, com as quais é possível haver comunicação; um líder social que é a melhor partícula na vizinhança $\vec{x}_{nbest}$, e o valor da função objetivo para essa posição ($\overrightarrow{x_{nbest}}$; $f(\vec{x}_{nbest})$); e um vetor que determina a velocidade dentro do espaço de busca $\vec{v}\ (t)$ [6].

Na fase de busca, cada partícula é atualizada pelas equações (3.1) e (3.2), fazendo com que a partícula se movimente pelo espaço de busca. A equação (3.1) combina uma atração para a memória cognitiva e a memória social, determinada pela vizinhança. Com isso, as partículas têm sua velocidade e seu posicionamento modificados. Esta atualização é realizada a cada iteração.

$$\vec{v}_i(t+1) = \omega\vec{v}_i(t) + c_1 r_1\left[\vec{x}_{pbest} - \vec{x}(t)\right] + c_2 r_2[\vec{x}_{nbest} - \vec{x}_i(t)]\ , \tag{3.1}$$

$$\vec{x}_i(t+1) = \vec{x}_i(t) + \vec{v}_i(t+1)\ , \tag{3.2}$$

onde $\vec{v}_i(t)$ é a velocidade da partícula *i* no instante *t*. c_1 e c_2 são coeficientes de aceleração cognitiva e coeficiente de aceleração social respectivamente. r_1 e r_2 são dois números aleatórios gerados por uma distribuição uniforme no intervalo [0,1]. ω é o coeficiente de inércia.

Na abordagem mais utilizada, as constantes de aceleração são iguais a 2,05 e o coeficiente de inércia é reduzido linearmente em função do número de iterações. A redução linear é calculada pela equação (3.3).

$$\omega = \left[(\omega_{max} - \omega_{min})\frac{k_{atual}}{k_{final}}\right], \tag{3.3}$$

onde k_{atual} é a iteração atual e k_{final} é o número total de iterações.

Ao longo das iterações, as mudanças de velocidade e de posição das partículas fazem estas se movimentarem dentro do espaço de busca do problema. Então, com influências individuais e sociais, o enxame de partículas acaba convergindo para uma solução ótima. Algumas variações interessantes que minimizam a dificuldade da abordagem original do PSO em manter a diversidade da população foram propostas recentemente.

5 Algoritmo APSO

O algoritmo PSO adaptativo (APSO) foi proposto por Zhang e colaboradores [8], pois foi identificado que o PSO original tinha uma baixa velocidade de convergência, além de uma capacidade limitada de escapar de mínimos locais. Nesta abordagem, os parâmetros do PSO são atualizados por um esquema sistemático baseados em regras *fuzzy*. Uma estratégia de aprendizagem elitista também foi empregada.

Os seguintes passos são executados a cada iteração do algoritmo:

- Avaliação da distribuição da população com a estimativa do estado evolucionário;
- Classificação da população baseada no estado evolucionário do enxame;
- Adaptar os parâmetros de aceleração, melhorando a velocidade de convergência;
- Execução da estratégia elitista como objetivo de escapar de mínimos locais;
- Adaptação do parâmetro de inércia para aperfeiçoar a busca.

5.1 Estimativa do Estado Evolucionário

O estado evolucionário do algoritmo é estabelecido através do fator evolucionário. Para calcular a estimativa do estado evolucionário primeiramente, faz-se necessário calcular a distância média de cada partícula para todas as outras com o auxilio da equação (4.1).

$$d_i = \frac{1}{N-1}\sum_{j=1,j\neq i}^{N}\sqrt{\sum_{k=1}^{D}(x_i^k - x_j^k)^2}, \tag{4.1}$$

em que N é número total de partículas no enxame e D é o número de dimensões do problema.

Subsequentemente, calcula-se o fator evolucionário pela equação (4.2).

$$f_{evol} = \frac{d_g - d_{min}}{d_{max} - d_{min}} \in [0,1], \tag{4.2}$$

onde f_{evol}é o fator evolucionário, d_g é a distância média entre a melhor partícula do enxame e as outras partículas, d_{min} é a menor distância média considerando todas as partículas do enxame e d_{max} é a maior distância média considerando todas as partículas do enxame.

5.2 Classificação da População

Para realizar a classificação do enxame em um dos estados evolucionários, foi proposto originalmente processos de classificação *fuzzy*. Funções de pertinência *fuzzy* são utilizadas para classificar o fator evolucionário. Existem vários estados possíveis, são eles: convergência, busca em profundidade, busca em amplitude e escape.

O estado de convergência ocorre quando o valor do fator evolucionário está próximo do mínimo. Neste caso a distância da melhor partícula do enxame para as demais é mínima. Assim o algoritmo busca boas soluções em uma região ótima.

Busca em profundidade ocorre quando o valor do fator evolucionário é pequeno, consequentemente, a distância da melhor partícula do enxame para as demais é pequena. Assim o algoritmo busca soluções em uma área limitada.

A busca em amplitude ocorre quando o valor do fator evolucionário atinge valores médios e altos. Neste caso, o algoritmo busca por uma região ótima.

Por fim, o estado de escape é atingido quando o valor do fator evolucionário é próximo do máximo. Isto significa que a melhor partícula do enxame escapou de um mínimo local para uma nova região de busca.

5.3 Adaptação dos parâmetros de aceleração

Inicialmente no APSO, os parâmetros de coeficientes de aceleração c_1 e c_2 têm seu valor definidos por 2,0. Durante a execução do algoritmo, esses coeficientes são atualizados de forma adaptativa em função do estado evolucionário como retrata a tabela 1.

Tabela 1 – Estratégia de Controle dos coeficientes de aceleração

Estado Evolucionário	c_1	c_2
Busca em Amplitude	Incrementar	Decrementar
Busca em Profundidade	Incrementar Levemente	Decrementar Levemente
Convergência	Incrementar Levemente	Incrementar Levemente
Escape	Decrementar	Incrementar

O incremento ou decremento dos coeficientes de aceleração é obtido através da taxa de aceleração (δ). Essa taxa é calculada em uma distribuição uniforme entre 0,05 e 0,01.

5.4 Estratégia Elitista

A estratégia elitista para aprendizado foi desenvolvida para ser aplicada apenas na melhor partícula do enxame. Assim ajudará a partícula a escapar de um mínimo local no estado de convergência.

É escolhida a dimensão da melhor partícula do enxame $\vec{x}_{pbest}(t)$. A esta dimensão será aplicada uma perturbação Gaussiana como mostra a equação (4.3).

$$\vec{x}_{pbest}(t+1) = \vec{x}_{pbest}(t) + (X^d_{max} - X^d_{min})G(\mu, \sigma^2), \tag{4.3}$$

em que (X^d_{max}, X^d_{min}) são os limites do espaço de busca, $G(\mu, \sigma^2)$ é um número aleatório obtido de uma distribuição gaussiana com média (μ) zero e desvio padrão (σ).

Esta nova posição somente será aceita se for melhor que a posição atual. Caso contrário, esta solução irá substituir a pior partícula do enxame.

5.5 Adaptação dos parâmetros de inércia

Para balancear a busca em amplitude e a busca em profundidade é realizada a adaptação do fator de inercia ω. O ajuste deste fator é calculado em função do fator evolucionário com o auxílio da equação (4.4).

$$\omega(f) = \frac{1}{1+1{,}5e^{-2{,}6f}} \in [0{,}4; 0{,}9],\ \forall f \in [0{,}1]\ , \tag{4.4}$$

6 Topologias para o PSO

A interação das partículas dentro do enxame pode ser comparada a uma rede populacional com conexões e membros sociais. Levando em conta esta analogia, a estrutura formada pelos membros pode ser definida como uma topologia. As partículas são membros que podem se comunicar com outras trocando informações que possibilitem o seu aprendizado. Como já foi comentado anteriormente, as partículas se movimentam influenciadas também pela comunicação com seus vizinhos. O fluxo de informações depende do grau de conectividade das partículas.

Em um enxame fortemente conectado, todas as partículas tenderão a executar a busca no mesmo local, fazendo com que o enxame convirja rapidamente com poucas iterações. Porém, existe a possibilidade de que todo o enxame fique preso em um único mínimo local. Já em topologias distribuídas, existe uma diminuição da possibilidade de ficar preso em mínimos locais, porém se faz necessário um alto número de iterações.

Várias topologias foram desenvolvidas com o objetivo de modificar o fluxo de informações no algoritmo PSO, como a estrela [9], von Neumann[9], Multi-ring [10], ClanPSO [11], ClanAPSO [6], entre outras.

7 Algoritmo ClanPSO

Uma nova topologia de partículas chamada ClanPSO foi proposta em [11] para melhorar o desempenho do Algoritmo PSO. Cada clã é formado por partículas que estão fortemente conectadas para o compartilhamento de informação.

A cada iteração, cada partícula do clã executa um processo de busca. Ao final da iteração, a partícula que obteve o melhor resultado de cada clã é eleita como líder. Então, um novo clã é formado somente com as partículas líderes de cada clã, e estas executam uma nova busca. Este processo é chamado de "Conferência dos Líderes". Na figura 1 pode ser observada a topologia ClanPSO em um exemplo na conferência dos líderes.

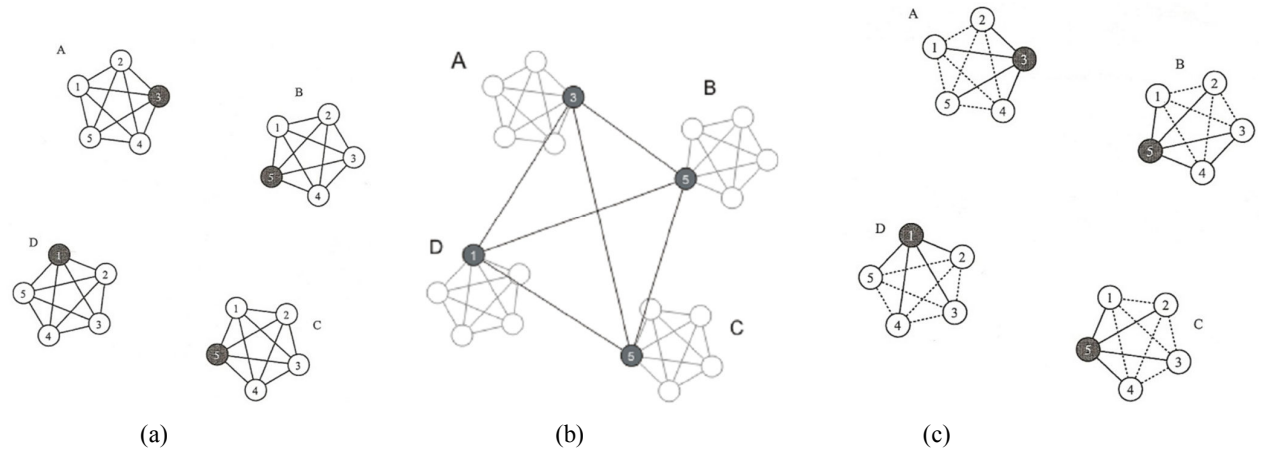

Figura 1 – Topologia ClanPSO. (a) delegação dos lideres em cada clã; (b) Conferência dos lideres; (c) Lideres disseminando informação dentro de seus clãs.

Ao final da conferência, cada líder retorna ao seu clã de origem e as informações adquiridas na conferência são usadas para atualizar as demais partículas. Como consequência, todas as partículas serão influenciadas indiretamente pela melhor posição encontrada pela melhor partícula do enxame inteiro.

8 Algoritmo ClanAPSO

O algoritmo ClanAPSO proposto por Pontes e colaboradores em 2011 [6] propôs incorporar o processo de adaptação sistemática de parâmetros em uma estratégia de busca com múltiplos enxames. A ideia é utilizar a topologia ClanPSO e executar o algoritmo *APSO* em cada um dos clãs. O pseudocódigo do algoritmo ClanAPSO está mostrado no algoritmo 1.

Algoritmo 1: Pseudocódigo do algoritmo ClanAPSO

1 inicializar o exame

2 **enquanto** o critério de parada não for atingido **faça**

3 **para** cada clã **faça**

4 execute o APSO dentro do *clan*;

5 selecione o líder;

6 **fim**

7 crie a conferência dos líderes;

8 execute o APSO com o líderes;

9 **fim**

9 Arranjo Experimental

Neste artigo, propõe-se integrar os algoritmos PSO, APSO, ClanPSO e ClanAPSO a uma rede neural do tipo MLP para que estes possam substituir o algoritmo *Backpropagation* como algoritmo de aprendizagem. Para isso, os algoritmos assumirão a tarefa de gerar e atualizar todos os pesos utilizados na rede neural, ou seja, cada partícula representará um vetor de pesos, os quais serão utilizados nas conexões entre os neurônios da rede.

A métrica adotada para o cálculo do *fitness* de cada partícula foi o erro médio quadrático (EMQ) da rede neural. Para cada partícula dos enxames existirá uma rede MLP que executará um ciclo com o vetor de pesos da partícula em todos os dados de treinamento e ao final, a rede retornará para a partícula o seu EMQ.

Para realizar os experimentos deste artigo foram utilizadas quatro bases de dados conhecidas que são usados para classificação [12]: Diagnostico de câncer de mama, que classifica em tumor benigno e tumor maligno baseado na descrição das células; Diagnostico de diabetes, que classifica o indivíduo em diabético ou não, a partir de dados pessoais e de resultados de exames médicos; Plantas Iris, que classifica a planta em três tipos (setosa, versicolor, virgínica) através de dados de largura e comprimento de pétala e sépala; Base Card, que avalia os dados de um indivíduo e classifica se o mesmo está apto ou não a receber o cartão de credito.

Para cada base de dados, foram realizados experimentos com uma rede MLP executando com o algoritmo de aprendizagem *Backpropagation*, PSO, APSO, ClanPSO, e por fim o ClanAPSO. Todas as redes foram configuradas com uma única camada escondida e esta contendo 20 neurônios. Além disso, foi utilizado o critério de validação cruzada como critério de parada de treinamento das redes. O critério para parada de treinamento foi idêntico em todos os experimentos, que são: máximo de iterações, constância do EMQ por 30 iterações consecutivas ou aumento do EMQ por 30 iterações consecutivas.

Para todas as bases, com cada um dos algoritmos de aprendizagem, foram realizadas 20 simulações e calculados sua média aritmética e desvio padrão. Os desvios padrão são apresentados entre parênteses.

10 Resultados

Inicialmente as simulações foram feitas com o algoritmo ClanAPSO. Três diferentes configurações foram testadas para definir a melhor, a qual será utilizada nas simulações subsequentes. A diferença nas configurações se deu pela quantidade de clãs e a quantidade de partículas em cada clã. O ClanAPSO 3X10 contém 3 clãs e cada um com 10 partículas. No caso do ClanAPSO 5X6, são 5 clãs com 6 partículas cada. E por fim, o ClanAPSO 10X3 que contém 10 clãs com 3 partículas cada.

As simulações com as diferentes configurações foram feitas nas 4 bases, onde observou-se a porcentagem de acerto, o tempo médio de treinamento em milissegundos e a quantidade de iterações média de treinamento. Na tabela 2, estão apresentados os resultados obtidos nas simulações das diferentes configurações do algoritmo ClanAPSO. Os valores representados na tabela representam a média aritmética das simulações e o que está entre parêntese representa o desvio padrão.

Os resultados marcados em negrito foram os melhores resultados obtidos em cada uma das bases. Com pode ser observado a configuração ClanAPSO 3X10 obteve melhor resultado, com relação a porcentagem de acerto, em três da quatros bases utilizadas e em todas as bases os testes com essa configuração obteve menor tempo de treinamento. Por isso, a configuração ClanAPSO 3X10 foi escolhida para ser utilizada nas próximas simulações.

Tabela 2 - Resultados obtidos nos experimentos com diferentes configurações do ClanAPSO (Acerto – porcentagem de acerto, Tempo – tempo gasto no treinamento em milissegundos, Iterações – quantidade de iterações de treinamento).

Câncer				**Card**			
Algoritmo	**Acerto(%)**	**Tempo(ms)**	**Iterações**	**Algoritmo**	**Acerto(%)**	**Tempo(ms)**	**Iterações**
ClanAPSO 3X10	95,79 (1,11)	**21272,55 (6487,95)**	541,45 (122,59)	**ClanAPSO 3X10**	**86,27 (2,59)**	**25558,9 (1913,37)**	599,4 (2,61)
ClanAPSO 5X6	95,99 (1,42)	24801,55 (7176,61)	525,9 (122,46)	**ClanAPSO 5X6**	85,14 (1,66)	28830,95 (1849,00)	599,4 (2,61)
ClanAPSO 10X3	**96,30 (1,39)**	25678,25 (5966,18)	557,05 (102,75)	**ClanAPSO 10X3**	85,31 (2,56)	31397,8 (5034,69)	582,55 (76,06)
Iris				**Diabetes**			
Algoritmo	**Acerto(%)**	**Tempo(ms)**	**Iterações**	**Algoritmo**	**Acerto(%)**	**Tempo(ms)**	**Iterações**
ClanAPSO 3X10	**96,71 (2,74)**	**4546,75 (937,78)**	569,45 (77,85)	**ClanAPSO 3X10**	**74,21 (2,41)**	**17192,6 (8183,45)**	419,35 (144,01)
ClanAPSO 5X6	94,73 (3,72)	5266,05 (873,62)	581,45 (58,54)	**ClanAPSO 5X6**	71,22 (4,69)	19539,65 (10132,52)	410,05 (175,77)
ClanAPSO 10X3	94,73 (3,00)	5138,65 (1215,46)	548,35 (111,78)	**ClanAPSO 10X3**	72,81 (2,88)	18026,05 (8485,36)	395,8 (165,19)

A tabela 1 mostra os resultados obtidos com os experimentos para todos os algoritmos de aprendizagem utilizados. Para realizar a análise de desempenho da rede foi observada a porcentagem de acerto de classificação, a quantidade de iterações média e o tempo médio necessários para o treinamento. Os resultados marcados em negritos são os melhores resultados obtidos.

Para a base Câncer, o melhor resultado de classificação foi obtido pelo algoritmo APSO. Apesar do ClanAPSO não ter obtido o melhor resultado nesta base, ele obteve resultado melhor que o *Backpropagation*. Na base *Card*, o melhor resultado foi alcançado pelo ClanAPSO, neste caso o APSO obteve resultado inferior ao ClanAPSO, porém melhor que o *Backpropagation*. Já na base Iris, somente o ClanAPSO obteve melhor resultado que o *Backpropagation,* mas o APSO não ficou muito longe, uma diferença de apenas aproximadamente 1,7%. E por fim, na base Diabetes, o ClanAPSO e o *Backpropagation* obtiveram resultados muito próximo, mas o ClanAPSO obteve um resultado ligeiramente melhor.

Tabela 3 - Resultados obtidos nos experimentos (Acerto – porcentagem de acerto, Tempo – tempo gasto no treinamento em milissegundos, Iterações – quantidade de iterações de treinamento).

Câncer				**Card**			
Algoritmo	**Acerto(%)**	**Tempo(ms)**	**Iterações**	**Algoritmo**	**Acerto(%)**	**Tempo(ms)**	**Iterações**
BackPropagation	95,26 (0,56)	697,45 (98,61)	127,5 (18,13)	**BackPropagation**	83,42 (0,86)	253,15 (487,16)	147,4 (51,86)
PSO	94,26 (2,58)	5557,65 (1621,88)	132,05 (33,76)	**PSO**	76,76 (6,36)	7988,5 (2683,80)	160,4 (46,92)
APSO	**96,64 (1,40)**	12811,55 (5294,62)	431,85 (151,58)	**APSO**	83,72 (3,80)	12247,45 (1169,24)	433,15 (182,59)
ClanPSO	95,93 (1,59)	7963,95 (3551,37)	80,4 (46,58)	**ClanPSO**	77,16 (7,19)	39390,15 (7518,68)	270,8 (53,68)
ClanAPSO 3X10	95,79 (1,11)	21272,55 (6487,95)	541,45 (122,59)	**ClanAPSO 3X10**	**86,27 (2,59)**	25558,9 (1913,37)	599,4 (2,61)
Iris				**Diabetes**			
Algoritmo	**Acerto(%)**	**Tempo(ms)**	**Iterações**	**Algoritmo**	**Acerto(%)**	**Tempo(ms)**	**Iterações**
BackPropagation	95,70 (1,68)	319,6 (31,32)	282,3 (18,30)	**BackPropagation**	**74,13 (1,35)**	1384,3 (17,41)	41,95 (2,06)
PSO	78,94 (8,72)	1807,15 (510,17)	138,9 (37,03)	**PSO**	66,17 (1,98)	5798,45 (1638,90)	116,3 (25,39)
APSO	94,07 (3,61)	3514,95 (1169,24)	450,25 (174,74)	**APSO**	70,02 (5,33)	7864,8 (6338,91)	251,45 (173,35)
ClanPSO	70,52 (10,21)	2562,3 (1135,03)	103,85 (51,66)	**ClanPSO**	65,28 (3,41)	25856,2 (3776,11)	248,8 (33,61)
ClanAPSO 3X10	**96,71 (2,74)**	4546,75 (937,78)	569,45 (77,85)	**ClanAPSO 3X10**	**74,21 (2,41)**	17192,6 (8183,45)	419,35 (144,01)

11 Conclusões

Este artigo teve como principal objetivo testar técnicas alternativas para realizar o processo de treinamento de Redes MLPs utilizadas para classificação. Foi mostrado que o algoritmo de treinamento comumente utilizado, algoritmo *Backpropagation*, apesar de seus pontos fracos, apresentou um bom desempenho nessas bases de dados. Analisando os resultados obtidos pelas diferentes técnicas aqui propostas, pode-se verificar que a taxa de acerto foi maior que a do algoritmo *backpropagation* em todas as simulações. O algoritmo ClanAPSO obteve melhor resultado que o *backprogation* em todas as bases. Já o algoritmo APSO mostrou-se melhor que o *backpropagation* nas bases Câncer e Card e ficou muito próximo na base Iris. Isso nos leva a perceber que os algoritmos adaptativos são uma boa alternativa ao algoritmo *backprogation.* Os melhores resultados foram obtidos para múltiplos sub-enxames. Desta forma, os resultados obtidos com os sistemas híbridos aqui propostos e testados são bastante promissores, principalmente o algoritmo ClanAPSO.

Apesar dos melhores resultados obtidos em relação ao *backpropagation*, o tempo de treinamento e a quantidade de iterações necessárias para a convergência dos algoritmos adaptativos tiveram um aumento considerável.

Nos próximos trabalhos pretende-se testar alternativas para a configuração dos parâmetros do algoritmo ClanAPSO, testar o desempenho destes algoritmos em problemas de previsão de séries temporais com alta dimensionalidade (grande número de entradas e saídas e grande número de exemplos), onde o algoritmo *Backpropagation* apresenta suas maiores dificuldades. Outra aplicação pode ser em problemas de otimização multi-objetivo, onde também se sabe que o algoritmo *Backpropagation* não apresenta bom desempenho.

12 Referências

[1] A. BRAGA, A. CARVALHO & T. LUDERMIR. **Redes Neurais Artificiais: Teoria e Aplicações**. Rio de Janeiro: LTC, 2000. 262 p.

[2] M. VALENÇA. **Fundamentos das Redes Neurais**. 2 ed. Olinda: Livro Rápido, 2009. 384 p.

[3] M. CARVALHO. **Uma Análise de Otimização de Redes Neurais MLP por Enxame de Partículas**. 2007. 66 f. Dissertação de Mestrado do Curso de Ciências da Computação, Universidade Federal de Pernambuco, Recife.

[4] P. SAITO JUNIOR, C. NASCIMENTO JUNIOR & T. YONEYAMA. **Treinamento de Redes Neurais Artificiais Utilizando Time Assíncrono**. In: IV Congresso Brasileiro De Redes Neurais, 1999, São José dos Campos. p. 078-083.

[5] D. T. PHAM, E. KOÇ, A. GHANBARZADEH & S. OTRI. **Optimisation of the Weights of Multi-Layered Perceptrons Using Bees Algorithm**. In: Proceedings of 5th International Symposium On Intelligent Manufacturing Systems, 2006, Sakarya University. p. 38-46.

[6] M. R. PONTES, F. B. LIMA-NETO & C. J. A. BASTOS-FILHO. **Adaptative Clan Swarm Optimization**. In: IEEE Symposium Series on Computational Intelligence - SSCI, 2011, Paris, França. Proceeding of IEEE Swarm Intelligence Symposium – SSCI 2011, 2011. p. 17-22.

[7] J. KENNEDY & R. C. EBERHART, **Particle swarm optimization**, in Proceeding of IEEE International Conference on Neural Networks, vol. 4, 1995.

[8] Z. ZHAN, J. ZHANG, Y. LI & H. CHUNG. **Adaptive particle swarm optimization**, IEEE Transactions on Systems Man, and Cybernetics-Part B: Cybernetics, vol. 39, no. 6, pp. 1362–1381, 2009.

[9] J., Kennedy & R., Mendes, **Particle Swarm Optimization**, Proceedings of the IEEE International Conference on Neural Networks, pp. 38-44.

[10] C. J. A. BASTOS-FILHO, D. F. CARVALHO, M. P. CARACIOLO, P. B. C. MIRANDA & E. M. N. FIGUEIREDO. **Multi-Ring Particle Swarm Optimization,** Evolutionary Computing, pp. 572, I-Tech, Vienna, Austria.

[11] D. F. CARVALHO & C. J. A. BASTOS-FILHO. **Clan particle swarm optimization**, International Journal of Intelligent Computing and Cybernetics, vol. 2, p. 1, 2009.

[12] D. AHA, A. ASUNCION & D. NEWMAN. **The UCI Machine Learning Repository**. Disponível em: <http://archive.ics.uci.edu>. Acesso em: 12/06/2011.