TREINAMENTO DE UMA REDE PERCEPTRON DE MÚLTIPLAS CAMADAS UTILIZANDO *ENHANCED-CONTINUOUS-GRASP*

Tiago Maritan Ugulino de Araújo[1], Jefferson Ferreira de Araújo Lima[1], Lucídio dos Anjos Formiga Cabral[1], Guido Lemos de Souza Filho[1] e Adrião Duarte Dória Neto[2]

[1]Laboratório de Aplicações em Vídeo Digital, Departamento de Informática, Universidade Federal da Paraíba, Cidade Universitária, CEP 58050-900, João Pessoa, Paraíba

[2]Departamento de Computação e Automação, Universidade Federal do Rio Grande do Norte, Lagoa Nova, CEP 59072-970, Natal, Rio Grande do Norte

maritan@lavid.ufpb.br; jefferson@lavid.ufpb.br; lucidiocabral@gmail.com; guido@lavid.ufpb.br; adriao@dca.ufrn.br

Resumo – O treinamento supervisionado de uma rede Perceptron de Múltiplas Camadas (*Multilayer Perceptron* - MLP) [1] baseado nos algoritmos baseados em gradiente, como o algoritmo de retropropagação do erro (*error backpropagation*) [2], geralmente possui algumas limitações como, por exemplo, a possibilidade de ficar preso em mínimos locais e a elevada complexidade computacional quando o número de parâmetros da rede (pesos sinápticos e bias) é grande [3]. Outra alternativa comumente explorada para o treinamento de MLP é a utilização de metaheurísticas para otimização combinatória, como Algoritmos Genéticos [4] e Busca Tabu [5]. Segundo Sexton e Guppa [6], contudo, essas abordagens são baseadas num esquema de codificação binária dos parâmetros da rede e não trazem necessariamente nenhum benefício. Para contornar esses problemas, nesse artigo propomos a utilização da metaheurística *Enhanced Continuous-GRASP* (EC-GRASP) [7,8], uma metaheurística para resolução de problemas de otimização global contínua, para o treinamento de uma rede MLP. Para validar o método proposto, a rede MLP foi treinada para aproximação de algumas funções não-lineares. Os resultados computacionais atestam a aplicabilidade da nova abordagem proposta.

Palavras-chave – Perceptron de Múltiplas Camadas, Treinamento Supervisionado, EC-GRASP, Metaheurísticas, Otimização Global Contínua, Redes Neurais Artificiais

1. Introdução

O treinamento supervisionado de uma rede Perceptron de Múltiplas Camadas (*Multilayer Perceptron* - MLP) [1], em geral, procura encontrar os valores dos parâmetros da rede (pesos sinápticos e bias) que minimizam o erro médio quadrático (E_{med}) associado a um conjunto de amostras de treinamento. O problema então pode ser visualizado como um problema de otimização irrestrito, definido matematicamente da seguinte forma:

$$\min E_{med}, \text{ onde} \tag{1}$$

$$E_{med} = \frac{1}{2N}\sum_{n=1}^{N}[d(n) - o(n,\mathbf{w})]^2 \tag{2}$$

Nesse contexto, E_{med} representa a *função de custo*, ou seja, a média dos erros quadráticos associados ao conjunto de N amostras de treinamento. O erro quadrático de cada amostra de treinamento n é definido como a diferença entre a a resposta desejada (ou saída alvo) $d(n)$ e a resposta real da rede ao padrão de treinamento $o(n,\mathbf{w})$. A resposta da rede $o(n,\mathbf{w})$ é determinada com base na amostra de treinamento n e nos parâmetros da rede $\mathbf{w}$.

Por serem métodos de busca exatos, os algoritmos baseados em gradiente para o treinamento de um MLP, como o algoritmo de retropropagação do erro (*error backpropagation*) [2], geralmente possuem algumas limitações como a possibilidade de ficar preso em mínimos locais e o crescimento exponencial da complexidade computacional na medida em que o número de parâmetros da rede (número de sinapses) aumenta [3]. Outra alternativa comumente explorada para o treinamento de MLP é a utilização de metaheurísticas para otimização combinatória, como, por exemplo, Algoritmos Genéticos [4] ou Busca

Tabu [5]. Segundo Sexton e Gupta [6], contudo, essas abordagens são baseadas num esquema de codificação binária dos parâmetros da rede e não trazem necessariamente nenhum benefício.

Para contornar esses problemas, nesse artigo, propomos a utilização do *Enhanced Continuous-GRASP* (EC-GRASP) [7,8], uma metaheurística projetada para resolução de problemas de otimização global contínua, para o treinamento de uma rede MLP. O EC-GRASP, uma adaptação da metaheurística *Greedy Randomized Adaptive Search Procedure* (GRASP) [9] para resolução de problemas de otimização global contínua, é um algoritmo simples de implementar, não utiliza derivadas da função objetivo e tem a capacidade de escapar de mínimos locais [8].

Esse artigo está organizado da seguinte forma. Na seção 2 descrevemos a metaheurística EC-GRASP. Na seção 3 apresentaremos como o EC-GRASP foi aplicado ao treinamento de uma rede MLP. Na seção 4 apresentamos os experimentos computacionais utilizados para validar o método proposto. Por fim, as conclusões são apresentadas na seção 5.

2. *Enhanced Continuous-GRASP* (EC-GRASP)

O EC-GRASP, uma evolução da metaheurística *Continuous-GRASP* (C-GRASP) [10, 11], foi uma alternativa proposta por [8] para resolução de problemas de otimização global contínua sujeito a restrições no domínio do problema (*box constraints*). Sem perda de generalidade, define-se o domínio S como sendo um hiper-retângulo $S = \{\mathbf{x} \in \Re^n \mid \mathbf{l} \leq \mathbf{x} \leq \mathbf{u}\}$, onde $\mathbf{l} \in \Re^n$, $\mathbf{u} \in \Re^n$ e $u_i \geq l_i$, para $i = 1,2,...,n$. O problema considerado, portanto, é:

$$\min f(\mathbf{x}), \tag{3}$$

$$\text{sujeito a } \mathbf{l} \leq \mathbf{x} \leq \mathbf{u}, \tag{4}$$

Onde $f : \Re^n \rightarrow \Re$ e $\mathbf{l}, \mathbf{x}, \mathbf{u} \in \Re^n$.

De forma similar ao C-GRASP [11], o EC-GRASP é um método de busca *multi-start*[1] que utiliza uma fase de construção aleatória e gulosa[2] para gerar as soluções iniciais e uma fase de busca local para tentar melhorar as soluções geradas pelo procedimento de construção. O EC-GRASP discretiza o domínio do problema em uma grade de pontos uniformemente distribuídos. Os procedimentos de construção e busca local realizam seus movimentos através dessa grade de pontos e, à medida que o algoritmo progride, essa grade vai adaptativamente se tornando mais densa. Na Figura 1 é apresentado o pseudocódigo do método EC-GRASP.

De acordo com a Figura 1, observa-se que o EC-GRASP possui como parâmetros de entrada: a dimensão do problema *n*; os vetores **l** e **u** que representam os limites inferiores e superiores do domínio de entrada, respectivamente; a função objetivo *f*(.); e os parâmetros h_s, h_e e MaxLocalIters. Os parâmetros h_s e h_e representam a densidade inicial e final da grade de discretização, respectivamente, e MaxLocalIters representa o número máximo de iterações sem melhoria que devem ser aplicadas durante o procedimento de busca local.

O algoritmo processa as iterações enquanto algum critério de parada não é satisfeito. Em cada iteração, inicialmente, um ponto aleatório uniformemente distribuído sobre os limites **l** e **u** é atribuído a solução atual **x** e o parâmetro *h*, que representa o tamanho atual do passo de discretização, é atualizado para o valor h_s. Em seguida, os procedimentos de construção e busca local são chamados seqüencialmente, em ciclo, partindo da solução atual **x**. Se o ciclo de construção-busca local não conseguir gerar uma solução melhor que a solução de entrada **x** (ou seja, as variáveis de estado ImprC e ImprL estão no estado **false**), o EC-GRASP aumenta a densidade da grade de discretização atual, reduzindo o valor de *h* (i.e., $h \leftarrow h/2$). Uma iteração do EC-GRASP termina quando $h < h_e$. A melhor solução visitada em todas as iterações do algoritmo (**x***) é retornada pelo algoritmo.

O EC-GRASP utiliza o mesmo procedimento de construção (Construção) do C-GRASP (definido em [11]) e define um novo procedimento de busca local, o procedimento BuscaLocal_NAPS. Esse

---

[1] Um procedimento *multi-start* é um procedimento iterativo que utiliza vários pontos de partida gerados de forma aleatória. Essa característica permite que o procedimento diversifique a busca no espaço de soluções, escapando de ótimos locais

[2] "Um algoritmo guloso é aquele que faz a escolha que parece melhor no momento. Isto é, ele faz uma escolha ótima para as condições locais, na esperança de que essa escolha leve a uma solução ótima para a situação global" [13]

procedimento é baseado no método *New Adaptive Pattern Search* (NAPS), um método de busca direcionada. O papel do NAPS é estabilizar a busca na vizinhança do mínimo local. Mais especificamente, ao invés de realizar movimentos aleatórios na vizinhança, uma estratégia de busca direcionada é responsável por gerar esses movimentos.

```
procedure EC-GRASP(n, l, u, f(.), h_s, h_e, MaxLocalIters)
1.    f* ← ∞;
2.    while Critério de parada não satisfeito do
3.        x⃗ ← UnifRand(l⃗, u⃗);
4.        h ← h_s;
5.        while h < h_e do
6.            ImprC ← false;
7.            ImprL ← false;
8.            [x⃗, ImprC] ← Construcao(x⃗, f(.), n, l⃗, u⃗, ImprC)
9.            [x⃗, ImprL] ← BuscaLocal_NAPS(x⃗, f(.), n, h, h_e
                           l⃗, u⃗, MaxLocalIters, ImprL);
10.           if f(x⃗) < f* then
11.               x⃗* ← x⃗;
12.               f* ← f(x⃗);
13.           end if
14.           if ImprC = false and ImprL = false then
15.               h ← h/2;
16.           end if
17.       end while
18.   end while
19. return x⃗*;
end EC-GRASP.
```

Figura 1 – Pseudocódigo do método EC-GRASP.

2.1. Procedimento de Construção

O procedimento de construção Construção do EC-GRASP é o mesmo procedimento utilizado pelo C-GRASP. Ele inicia permitindo que todas as coordenadas da solução de entrada **x** possam ser alteradas, criando um conjunto de coordenadas denominadas coordenadas não-fixas (Unfixed). A cada iteração, é efetuada uma busca linear na direção de cada coordenada não-fixa de **x** com as outras (n-1) coordenadas mantidas com os seus valores atuais. Uma Lista de Candidatos Restrita (*Restricted Candidate List* - RCL) com tamanho determinado por $\alpha \in [0,1]$ é criada apenas com as coordenadas não-fixas. Um elemento da RCL é então escolhido aleatoriamente e essa coordenada é retirada do conjunto de coordenadas não-fixas. O procedimento é repetido até que o conjunto de coordenadas não-fixas esteja vazio. O pseudocódigo do procedimento Construção é apresentado na Figura 2.

2.2. Método *New Adaptive Pattern Search* (NAPS)

O método *New Adaptive Pattern Search* (NAPS) é um método baseado no método *Adaptive Pattern Search* (APS) proposto em [12]. Ele é um procedimento livre de derivadas com uma grande habilidade de gerar direções de descida utilizando soluções ao redor da solução atual. Mais especificamente, o NAPS constrói *n* direções paralelas aos eixos coordenados, partindo de uma solução atual **x**, e gera um conjunto $T = \{\mathbf{y}\}_{i=1}^{n}$ de *n* pontos ao longo dessas direções com um tamanho de passo *h*. A direção adaptativa **d** é calculada da seguinte forma:

$$\mathbf{d} = \sum_{i=1}^{n} \omega_i \mathbf{u}_i \text{, onde,} \qquad (5)$$

$$\omega_i = \frac{\Delta f_i}{\sum_{j=1}^{n} |\Delta f_j|}, \quad i = 1,2,\ldots,n, \tag{6}$$

$$\mathbf{u}_i = -\frac{(\mathbf{y}_i - \mathbf{x})}{\|(\mathbf{y}_i - \mathbf{x})\|}, \quad i = 1,2,\ldots,n, \tag{7}$$

$$\Delta f_i = f(y_i) - f(x), \quad i = 1,2,\ldots,n \tag{8}$$

Após calcular a direção de descida, aplica-se uma busca linear nessa direção (a direção **d**). A solução retornada pela busca linear $\mathbf{y}_{n+1}$ é então adicionada ao conjunto T e o melhor ponto de T é retornado pelo método NAPS. O pseudocódigo do procedimento NAPS é apresentado na Figura 3.

```
procedure Construção(x⃗, f(.), n, l⃗, u⃗, ImprC)
1.    Unfixed = {1, 2, ..., n};
2.    α ← UnifRand(0, 1);
3.    ReUse ← false;
4.    while Unfixed ≠ ∅ do;
5.        g_min = +∞; g_max = −∞;
6.        for each i ∈ Unfixed do
7.            if ReUse= false then
8.                z_i ← LinearSearch(x⃗, f(.), h, i, n, l⃗, u⃗);
9.                g_i ← f(x̂^i);
10.           end if
11.           if g_min > g_i then g_min ← g_i;
12.           if g_max < g_i then g_max ← g_i;
13.       end for
14.       RCL ← ∅;
15.       Threshold ← g_min + α(g_max − g_min);
16.       for each i ∈ Unfixed and g_i ≤ Threshold do
17.           RCL ← RCL ∪ i;
18.       end for
19.       j ← RandomSelectElement(RCL);
20.       if x_j = z_j then ReUse ← true;
21.       else
22.           x_j ← z_j;
23.           ReUse ← false;
24.           ImprL ← true;
25.       end if
26.       Unfixed ← Unfixed\{j};
27.   end while
28. return [x⃗, ImprC];
end Construction
```

Figura 2 – **Pseudocódigo do procedimento de construção do EC-GRASP.**

2.2. Procedimento de Busca Local

O procedimento de busca local do EC-GRASP (BuscaLocal_NAPS) é um método iterativo de busca local que não utiliza cálculos de derivada e explora a vizinhança da solução atual aceitando apenas soluções melhores que ela. A vizinhança do procedimento de busca local é baseada no método NAPS

(descrito na seção 2.2) e na vizinhança $B_h(\mathbf{x}^*)$, definida em [11]. O procedimento BuscaLocal_NAPS, em cada iteração, calcula uma direção de descida aproximada **d**, a partir da solução atual **x** e realiza uma busca linear nessa direção (procedimento NAPS). Se a solução retornada pelo NAPS não for melhor que a solução **x*** (a melhor solução visitada até o momento), o procedimento BuscaLocal_NAPS atualiza a solução corrente **x** com uma solução escolhida aleatoriamente do conjunto $B_h(\mathbf{x}^*)$. O procedimento termina quando um determinado número de tentativas sem melhorias, na solução **x***, é extrapolado. Quando isso acontece, o procedimento retorna a solução **x***. O pseudocódigo do procedimento BuscaLocal_NAPS é apresentado na Figura 4.

```
procedure NAPS(x⃗, f(.), n, h, h_e)
1. Gera o conjunto T ← {y⃗}_{i=1}^{n} usando x⃗ e h;
2. for i = 1, ..., n do
3.     if f(y⃗_i) < f(x⃗) then return y⃗_i;
4. end for
5. Calcula d usando (5);
6. y_{n+1} ← LinearSearch(x⃗, f(.), d, h, h_e);
7. y_min ← min(y_1, ..., y_{n+1});
8. return y_min;
end NAPS.
```

Figura 3 – Pseudocódigo do procedimento NAPS.

```
procedure BuscaLocaLNAPS (x⃗, f(.), n, h, h_e, l⃗, u⃗,
                          MaxLocalIters, ImprL)
1. x⃗* ← x⃗;
2. f* ← f(x⃗*);
3. NumIters ← 0;
4. while NumIters ≤ MaxLocalIters do
5.     NumIters ← NumIters + 1;
6.     x⃗ ← NAPS(x⃗, f(.), n, h, h_e);
7.     if l⃗ ≤ x⃗ ≤ u⃗ and f(x⃗) < f* then
8.         x⃗* ← x⃗;
9.         f* ← f(x⃗);
10.        ImprL ← true;
11.        NumIters ← 0;
12.    else
13.        x⃗ ← RandomlySelectElement(B_h(x⃗*));
14.    end if
15. end while
16. return [x⃗*, ImprL];
end BuscaLocaLNAPS.
```

Figura 4 – Pseudocódigo do procedimento de busca local do EC-GRASP.

3. Treinando um MLP com EC-GRASP

Conforme já mencionado na seção 2, para aplicar o EC-GRASP ao treinamento de um MLP, é necessário definir um par de vetores **l** e **u** que definem o domínio de entrada do problema (ou seja, os valores possíveis dos pesos sinápticos). Esse problema pode ser resolvido definindo vetores **l** e **u** para

cada problema. Nesse caso, o problema de treinar uma rede MLP com o EC-GRASP pode ser definido da seguinte forma:

$$\min E_{med}, \text{ onde} \tag{9}$$

$$E_{med} = \frac{1}{2N}\sum_{n=1}^{N}[d(n) - o(n, \mathbf{w})]^2, \tag{10}$$

$$\text{sujeito a } \mathbf{l} \le \mathbf{x} \le \mathbf{u}, \tag{11}$$

Onde E_{med} representa o erro médio quadrático sobre todas as N amostras de treinamento; $\mathbf{w}$ representa os parâmetros da rede; $d(n)$ e $o(n, \mathbf{w})$ representam a resposta deseja e a resposta real da rede a amostra de treinamento n, respectivamente; $\mathbf{l}$ e $\mathbf{u}$ representam o domínio de entrada, ou seja, o intervalo de valores que o vetor $\mathbf{w}$ pode assumir.

4. Experimentos Computacionais

Para validar o método proposto, utilizou-se o EC-GRASP para treinar duas redes MLP com o objetivo de aproximar as seguintes funções não-lineares abaixo:

- Função 1: $f(x) = 1/x$, onde $0.1 \le x \le 10$
- Função 2: $f(x_1, x_2) = x_1^2 + x_2^2 + 2x_1x_2 + x_1 + x_2 - 1$, onde $-1 \le \mathbf{x} \le 1$

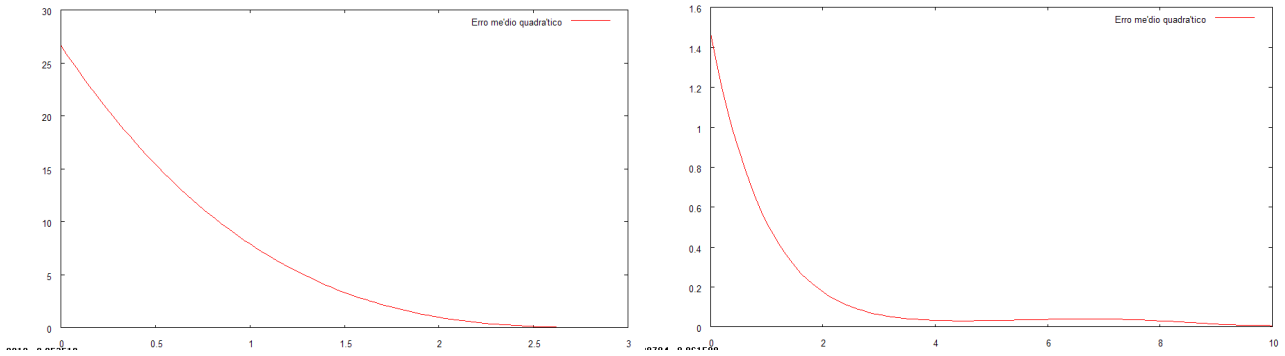
Na aproximação das funções 1 e 2 utilizou-se duas redes MLP com arquiteturas (definidas empiricamente) 1:4:1 e 2:4:2:1, respectivamente. Nas duas redes, as camadas ocultas eram formadas por neurônios com função de ativação tangente sigmóide e formadas por neurônios com função de ativação linear na camada de saída. Cada rede foi treinada com um conjunto de treinamento de 500 amostras geradas aleatoriamente no espaço de entrada.

O critério de parada do algoritmo EC-GRASP seria obter um erro médio quadrático inferior a 0.01 ($E_{med} < 0.01$) ou completar 20 iterações *multi-start*. O parâmetro n (dimensão do problema) é o número de parâmetros da rede (pesos sinápticos e bias) de cada rede MLP. Os demais parâmetros utilizados no EC-GRASP são apresentados na Tabela 1.

Tabela 1 – Parâmetros do EC-GRASP

Parâmetro	Valor	Parâmetro	Valor
$f(.)$	E_{med}	h_s	1.0
$\mathbf{l}$	$[-5,..,-5]^n$	h_e	0.1
$\mathbf{u}$	$[5,..,5]^n$	MaxLocalIters	$2n$

As Figuras 5(a) e 5(b) apresentam as curvas do erro médio quadrático de treinamento das Funções 1 e 2, respectivamente, em relação ao número de ciclos construção-busca local do EC-GRASP. De acordo com essas figuras, podemos observar que em poucos ciclos construção-busca local o EC-GRASP conseguiu convergir, ou seja, encontrar um erro médio quadrático menor que 0.01.

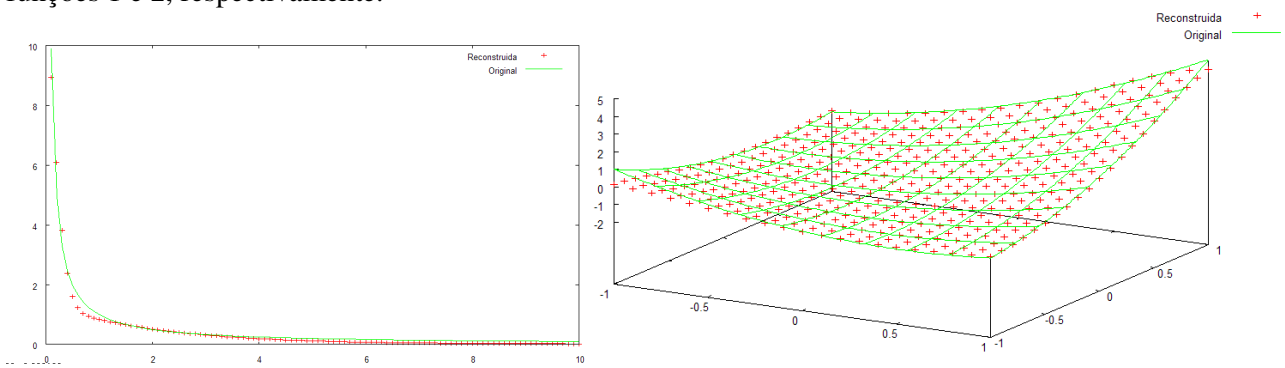


(a) (b)

Figura 5 – Erro médio quadrático versus número de ciclos "construção-busca local" do EC-GRASP para a (a) função 1 e (b) função 2.

Após o treinamento, as redes MLP foram validadas para um conjunto de validação formado por 500 amostras uniformemente distribuída no espaço de entrada. Nas Figuras 6(a) e 6(b) são apresentadas

as curvas originais confrontadas com a curva gerada pelas redes MLP treinadas com EC-GRASP para as funções 1 e 2, respectivamente.



(a) (b)

Figura 6 – Curva original versus curva reconstruída (a) da função 1 e (b) da função 2.

Dando continuidade aos experimentos computacionais, para as duas funções testadas, foi realizada uma comparação entre o esforço computacional exigido com o EC-GRASP e com o algoritmo da retropropagação do erro (*error backpropagation* - BP). Para isso, os dois algoritmos de treinamento foram aplicados nas duas redes MLP para 100, 1000 e 10000 épocas. Para uma comparação mais justa, utilizou-se o modo lote (*batch*) para o treinamento das redes MLP utilizando o algoritmo de retropropagação da rede. As Tabelas 2 e 3 apresentam os erros médios quadráticos na aplicação dos dois algoritmos de treinamento (EC-GRASP e BP) nas redes MLP utilizadas para aproximar as funções 1 e 2, respectivamente.

Tabela 2 – Erros médios quadráticos da rede MLP utilizada para aproximar a função 1

Algoritmo de treinamento	Número de épocas		
	100	1000	10000
EC-GRASP	0.048240	0.037096	0.019812
BP	0.000524	0.000016	0.000008

Tabela 3 – Erros médios quadráticos da rede MLP utilizada para aproximar a função 1

Algoritmo de treinamento	Número de épocas		
	100	1000	10000
EC-GRASP	0.678431	0.039308	0.038259
BP	0.000599	0.000035	0.000005

5. Conclusões

Nesse artigo, propomos a utilização da metaheurística EC-GRASP para o treinamento de uma rede perceptron de múltiplas camadas (MLP). Para validar o método proposto, foram treinadas duas redes MLP para aproximar duas funções não-lineares. Os resultados computacionais mostraram que as redes MLPs treinadas pelo EC-GRASP obtiveram um baixo erro médio quadrático para as funções testadas utilizando poucos ciclos de construção-busca local do EC-GRASP. Além disso, apesar de apresentar um desempenho inferior ao algoritmo de retropropagação do erro, para as funções testadas, o EC-GRASP obteve em todos os testes, um erro médio quadrático próximo de zero.

Referências:

[1] S. Haykin, **Redes Neurais: Princípios e prática**, Bookman, 2. ed., 2001.

[2] P. Werbos, **The Roots of the Backpropagation: From Ordered Derivatives to Neural Networks and Political Forecasting**, Wiley, 1993.

[3] M. N. H. Siddique, M. O. Tokhi, Training neural networks: backpropagation vs. genetic algorithms, **Proceedings of International Joint Conference on Neural Networks**, 4(2001), 2673-2678.

[4] A. C. M. L. Albuquerque, J. D. de Melo, A. D. D. Neto, Algoritmos Genéticos e Processamento Paralelo Aplicado ao Treinamento e Definição de Redes Neurais Perceptrons de Múltiplas Camadas, **Anais do XV Congresso Brasileiro de Automática**, 1(2004), 1-6.

[5] R. Battiti, G. Tecchiolli, Training neural nets with the reactive tabu search, **IEEE Transactions on Neural Networks**, 6(1995), 1185-1200.

[6] R. S. Sexton, J. N. D. Gupta, Comparative evaluation of genetic algorithm and backpropagation for training neural networks, **Information Sciences**, 129(2000), 45-59.

[7] T. M. U. Araújo, L. A. F. Cabral, R. Q. Nascimento, Hibridizando a metaheurística C-GRASP com o método de busca por padrões adaptativos para resolução de problemas de otimização global contínua, **Anais do XV Simpósio de Engenharia de Produção**, XV SIMPEP, 1(2008).

[8] T. M. U. Araújo, **Métodos híbridos baseados em Continuous-GRASP aplicados à otimização global contínua**, Master thesis, Universidade Federal da Paraiba, 2009.

[9] T. A. Feo, M. G. C. Resende, Greedy randomized adaptive search procedures, **Journal of Global Optimization**, 6(1995), 109-133.

[10] M. J. Hirsch *et al.*, Global optimization by continuous GRASP, **Optimization Letters**, 1(2007), 201-212.

[11] M. J. Hirsch *et al.*, A continuous GRASP to determine the relationship between drugs and adverse reactions, **AIP Conference Proceedings**, 1(2008), 106-121.

[12] A. R. Hedar, M. Fukushima, Tabu search directed by direct search methods for nonlinear global optimization, **European Journal of Operational Research**, 170(2006), 320-349.

[13] T. H. Cormen *et al.*, **Algoritmos: Teoria e Prática**, Editora Campus, 2. ed, 2002.