Aceleração do Algoritmo Competitivo Utilizando a Técnica ENNS: Aplicação em Projeto de Quantizadores Vetoriais de Imagens

Lucas Rocha Arruda
Ciência da Computação
Universidade Católica de Pernambuco
Recife, Pernambuco 50050-900
Email: lucasrarruda@gmail.com

Juliano Bandeira Lima
Departamento de Matemática
Universidade Federal de Pernambuco
Recife, Pernambuco 50740-560
Email: juliano@dmat.ufpe.br

Francisco Madeiro
Centro de Ciências e Tecnologia
Universidade Católica de Pernambuco
Recife, Pernambuco 50050-900
Email: madeiro@dei.unicap.br

***Resumo*—Este trabalho apresenta uma alternativa de aceleração de um algoritmo de aprendizagem não-supervisionada de redes neurais aplicado ao projeto de dicionários para quantização vetorial de imagem. A aceleração é obtida por meio do uso do algoritmo ENNS (*Equal-average Nearest Neighbor Search*) na etapa de determinação do neurônio vencedor. Resultados de simulações apresentam que a alternativa apontada leva a economias de cerca de 93,84% em termos de tempo gasto para projetar o dicionário.**

I. Introdução

Compressão de sinais (imagem e voz) é uma aplicação típica de quantização vetorial. Neste cenário, a técnica apresenta uma superioridade sobre a quantização escalar e permite obter elevadas taxas de compressão. A qualidade dos sinais comprimidos está diretamente relacionada à qualidade dos dicionários (quantizadores vetoriais) projetados. O algoritmo mais conhecido para elaboração de dicionários é o Algoritmo *Linde-Buzo-Gray* [1], também conhecido como GLA (*Generalized Lloyd Algorithm*). Outras abordagens têm sido usadas para projetar quantizadores vetoriais, como é o caso de algoritmos de aprendizagem não-supervisionada de redes neurais [2, 3, 4, 11, 13, 17], algoritmos Fuzzy [12, 16] e algoritmos meméticos [14, 15].

A Quantização Vetorial (QV) [5, 6] pode ser definida como um mapeamento *Q* de um vetor de entrada *x* pertencente ao espaço euclidiano *K*-dimensional, $\mathbb{R}^K$, em um vetor pertencente a um subconjunto finito *W* de $\mathbb{R}^K$, ou seja,

$$Q : \mathbb{R}^K \rightarrow W.$$

W é o dicionário e w_i são os vetores código, em que $i = 1, 2, ..., N$. O dicionário é um conjunto de vetores de reprodução (também chamados de vetores código), em que *K* é a dimensão do quantizador e *N* é o tamanho do dicionário (número de vetores código). O mapeamento *Q* cria o particionamento de $\mathbb{R}^K$ em *N* células (denominadas regiões de Voronoi) $S_i, i = 1, 2, ..., N$, tais que $S_i \cap S_j = \emptyset \quad \text{para} \quad i \neq j$. A versão quantizada de x é o vetor código de maior similaridade.

Um problema relevante em QV é a complexidade computacional da etapa de codificação. Diversas técnicas (e.g. [8, 18]) têm sido propostas para reduzir números de operações lógicas e aritméticas. Dentre as técnicas, uma que se destaca é o algoritmo ENNS [8, 10].

O presente trabalho contempla aplicações do ENNS em um cenário diferente: em projeto de dicionário por meio de aprendizagem competitiva. O trabalho mostra como o ENNS pode ser acomodado em um algoritmo de aprendizagem competitiva. Resultados de simulações mostram que, em projeto de dicionários aplicados à compressão de imagens, a acomodação do ENNS na etapa de determinação do vizinho mais próximo, permite reduzir cerca de 93,84% do tempo gasto para projetar o dicionário.

O trabalho encontra-se dividido da forma a seguir. A Seção II descreve brevemente o Algoritmo Competitivo. Na seção III, é discutido o uso do PDS (*Partial Distance Search*) e ENNS para reduzir o tempo de projeto de dicionário. Alguns detalhes da implementação dos algoritmos são brevemente descritos na Seção IV. Os resultados e os comentários finais são apresentados nas Seções V e VI, respectivamente.

II. Algoritmo Competitivo

O Algoritmo Competitivo (AC) é uma das técnicas que pode ser utilizada para o projeto de quantizadores vetoriais. O AC é descrito na Figura 1. No funcionamento do algoritmo, Competitivo, toda vez que um vetor de treino é comparado com todos os vetores código, apenas o mais semelhante, ou seja, o vencedor, tem suas componentes atualizadas. Na Figura 1, o n_{AC} denota o número total de iterações do algoritmo, $w_{i^*}(n, m)$ define o vencedor quando da apresentação do *m*-ésimo vetor de treino na *n*-ésima iteração, $w_i(n, m)$ é o *i*-ésimo vetor código e $x(m)$ é o *m*-ésimo vetor do conjunto de treino.

A atualização dos vetores código no AC é feita da seguinte maneira,

$$\tilde{w}_{i^*j}(n, m) = w_{i^*j}(n, m) + \eta(n)[x_j(m) - w_{i^*j}(n, m)],$$

em que $w_{i^*j}(n, m)$ é a *j*-ésima componente do vencedor, i^* denota o índice do vetor código com a menor distância encontrada em relação ao *m*-ésimo vetor de treino, $x_j(m)$

Algoritmo Competitivo (AC):

Para $1 \leq n \leq n_{\mathrm{AC}}$

Para $1 \leq m \leq M$

Determine o vencedor $\boldsymbol{w}_{i^*}(n,m)$:

$$i^* = \arg\min_i d[\boldsymbol{x}(m), \boldsymbol{w}_i(n,m)]$$

Atualize o vencedor de acordo com

$$\tilde{w}_{i^*j}(n,m) = w_{i^*j}(n,m) + \Delta w_{i^*j}(n,m),$$

em que

$$\Delta w_{i^*j}(n,m) = \eta(n)[(x_j(m) - w_{i^*j}(n,m)].$$

Figura 1. Descrição do Algoritmo Competitivo [2].

é a *j*-ésima componente do *m*-ésimo vetor de treino, $\eta(n)$ é a taxa de aprendizagem na *n*-ésima iteração do algoritmo ($0 < \eta(n) < 1$) e $\tilde{w}_{i^*j}(n,m)$ é a versão atualizada da *j*-ésima componente do vencedor. A taxa de aprendizagem $\eta(n)$ é definida como

$$\eta(n) = \eta(1) + (n-1)\frac{\eta(n_{AC}) - \eta(1)}{n_{AC} - 1},$$

em que $\eta(n_{AC})$ e $\eta(1)$ são, respectivamente, as taxas de aprendizagem final e inicial.

O Algoritmo Competitivo, originalmente, utiliza a busca total para encontrar os vetores código vencedores, ou seja, são realizadas operações aritméticas entre cada componente dos vetores código com cada componente dos vetores de treinamento. A cada iteração do AC, esse percorre todos os vetores de treino e, para cada um, pesquisa entre todos os vetores código, de forma sequencial, aquele que seja mais semelhante ao vetor de treino para ser atualizado. Para determinar a menor distância entre os vetores código e de treino, usa-se a distância euclidiana quadrática,

$$d[x(m), w_i(n,m)] = \sum_{j=1}^{K} [x_j(m) - w_{ij}(n,m)]^2,$$

em que *K* é a dimensão do dicionário.

III. Partial Distance Search

Na seção anterior, foi descrito brevemente o AC e também foi visto que o procedimento convencional de busca exaustiva (busca total) pelo neurônio vencedor envolve o cálculo de N distâncias (do vetor de treino para cada neurônio e o vetor código). Depois, determina-se a menor das N distâncias, o que requer $N-1$ comparações. Ressalta-se que cada cálculo de distância requer K multiplicações, K subtrações e $K-1$ adições. Originalmente o AC realiza uma busca total que consome um tempo de execução computacional que poderia ser reduzido, pois são realizados cálculos desnecessários. Um exemplo de cálculos desnecessários é descrito a seguir: caso um $w_i(n,m)$ qualquer seja calculado com uma determinada distância *D*, e, portanto, para um $j' < K$ o vetor código $w_{(i+1)}(n,m)$ já apresentar uma distância $D' > D$, é desnecessário processar as demais operações aritméticas envolvendo as componentes remanescentes de $w_{(i+1)}(n,m)$. Portanto, para economizar $K - j'$ subtrações, multiplicações e adições usa-se a técnica denotada como Partial Distortion Search (PDS). O PDS, descrito detalhadamente em [7], causa a saída prematura na busca pelo vetor código mais semelhante, o que reduz o tempo de busca do vencedor e, portanto, acelera o projeto do dicionário, diminuindo o tempo de execução do algoritmo em comparação ao uso da busca total. Contudo, deve-se observar que o número de comparações aumenta toda vez que se precisa verificar se é necessário calcular o restante do somatório.

IV. Uso do ENNS e PDS para acelerar o AC

Usando o AC com PDS, percebeu-se que alguns vetores código eram mais atualizados do que outros. Isto significa que alguns vencedores se repetiam mais do que outros. Portanto, sabe-se que se, no momento de realizar a busca do vetor código vencedor, com o uso da técnica do PDS, esse encontrar-se em primeiro na sequência de busca do vencedor, ou for um dos primeiros, o tempo de busca e o número de operações necessárias para determinar o vetor código mais semelhante ao vetor de treino será reduzido. Contudo, seria difícil manter sempre essa ordenação dos vetores código, ou seja, manter os vetores código que vencem com mais frequência em primeiro e os que vencem com menos frequência ao final. Essa dificuldade existe pelo fato de que, a cada vetor de treino, o vencedor é atualizado. Logo, os valores são alterados, exigindo uma nova ordenação. Analisando os custos de algoritmos de ordenação para manter estes vetores ordenados, verifica-se que não seria viável ordenar os vetores código depois de cada atualização, pois o tempo de projeto de dicionário poderia crescer.

Foi investigada uma técnica para manter essa ordenação sem elevar o custo computacional. Foi encontrada uma solução com um algoritmo chamado *Equal-average Nearest Neighbor Search* (ENNS). O ENNS é descrito em [8, 10], sendo usado originalmente no processo de codificação por distância mínima, da quantização vetorial. O ENNS usa o valor médio dos componentes de cada vetor código e vetor de treinamento, para comparar e, caso necessário, rejeitar vetores código distantes. Esse algoritmo reduz o tempo computacional em relação à busca total com apenas N (tamanho do dicionário) alocações de memória a mais, em relação ao PDS, fatos comprovados em [8, 10]. As N alocações a mais são justamente as médias calculadas de cada vetor código. O ENNS é mostrado com sucesso em relação à busca total e PDS, quando usado para codificação, mas, no projeto de dicionário, a quantidade de ordenações das médias precisa ser estabelecida de tal forma que mantenha a mesma eficiência de quando usado na codificação. Neste trabalho, foi investigada a possibilidade de ordenar os vetores código uma única vez, de tal forma que esta ordenação acelerasse o projeto de dicionário por meio do Algoritmo Competitivo. Essa escolha de ordenar uma única vez se dá pelo fato de os vetores código serem atualizados diversas

vezes a cada iteração. Logo, para manter uma ordenação constante das médias dos vetores código seria necessário ordenar os vetores a cada atualização de um determinado vetor código, o que teria um grande custo computacional. Para determinar o melhor momento para realizar essa única ordenação, deve-se levar em consideração alguns fatores. Um desses fatores é a qualidade da imagem reconstruída a cada iteração do projeto de dicionário (Figura 2). Note que, à medida que a imagem está melhor reconstruída, significa que os vetores código estão mais próximos, em termos de representação, dos vetores de treino. Levando isso em consideração, observa-se que vetores código mais próximos dos vetores de treino não sofreram variações que alterem tão significativamente suas médias, mas, ainda assim, a ordem dos vetores será afetada. Vale lembrar que a taxa de aprendizagem, $\eta(n)$, do Algoritmo Competitivo garante que a variação dos valores dos vetores código diminuem a cada iteração, variando entre a taxa de aprendizagem inicial, $\eta(1)$, e a final, $\eta(n_{AC})$. Tomando essas observações, podemos pesquisar qual seria o melhor momento para ser realizada a ordenação e então utilizar o ENNS a partir desse ponto. O uso do ENNS a partir da ordenação para encontrar o vetor código mais semelhante ao vetor de treino é justificado por não utilizar mais as operações aritméticas necessárias para a distância euclidiana para encontrar esse vetor código vencedor, e apenas comparações entre médias.

Na Tabela I, são mostrados os tempos de projeto de dicionário para o uso da ordenação, uma única vez e, portanto, a entrada do ENNS para encontrar os vetores código vencedores através da média. A Tabela I apresenta o tempo de projeto de dicionário para vários cenários do AC com ENNS. Cada cenário mostra o tempo necessário para o projeto de dicionário considerando o uso da ordenação em momentos distintos, como na primeira iteração, em seguida apenas na segunda e assim sucessivamente até na quinta iteração (os detalhes de implementação serão discutidos na próxima seção). Vemos, então, que o tempo de projeto do dicionário para a ordenação aplicada apenas na primeira iteração é menor do que as demais situações.

Tabela I
TEMPO DE EXECUÇÃO EM SEGUNDOS PARA PROJETAR O DICIONÁRIO (PARA N = 512) POR MEIO DO ALGORITMO COMPETITIVO COM ENNS, COM O USO DA ORDENAÇÃO E ENTRADA DO ENNS EM MOMENTOS DIFERENTES, NAS ITERAÇÕES 1, 2, 3, 4 OU 5.

Iteração	Tempo em Segundos
1	2,118
2	3,492
3	4,474
4	6,559
5	7,827

Neste artigo, foi utilizado o Algoritmo Competitivo com PDS junto ao algoritmo ENNS, aplicado ao projeto de dicionários. O PDS é associado ao AC para acelerar a busca do vencedor, enquanto que o ENNS tem a função de aperfeiçoar ainda mais a busca do vencedor ordenando os vetores código. Na primeira iteração, usa-se o PDS para atualizar o dicionário.

Figura 2. Variação da qualidade da imagem em função do número de iterações: a) 1 iteração, b) 2, c) 3 e d) 4 iterações.

Da segunda iteração em diante, os vetores são ordenados (uma única vez, como já discutido) e, então, a partir desse ponto, é utilizado o algoritmo ENNS (com o PDS incorporado dentro do funcionamento do ENNS, como em [8]) para ordenar os vetores código e tornar a busca dos vencedores mais eficiente. Na segunda iteração, pode-se ordenar os vetores, apenas esta vez (ordenar os vetores mais vezes aumentaria o tempo de execução do algoritmo deixando-o maior, em relação ao Algoritmo Competitivo na forma original). Dessa maneira, pretende-se reduzir ainda mais o número de operações aritméticas, já que com o ENNS não há necessidade de usar constantemente a distância euclidiana para determinar o vencedor, mas apenas a média, que é calculada uma única vez. Portanto, o funcionamento do AC com ENNS ficará da seguinte forma: na primeira iteração, para cada vetor de treino, é pesquisado o respectivo vencedor utilizando a distância euclidiana com o PDS; a partir da segunda iteração, calcula-se a média de todos os vetores código e de treino para encontrar os vetores código vencedores, utilizando o ENNS na atual e posteriores iterações.

V. DETALHES DE IMPLEMENTAÇÃO E SIMULAÇÃO

Vale observar alguns detalhes de implementação do ENNS, como o uso do algoritmo *Heapsort* [9] para ordenar os vetores após o cálculo da média, da mesma forma que é necessário o uso da pesquisa binária [9] para fazer a busca do vetor código vencedor. A escolha do *Heapsort* como mecanismo de ordenação é justificada por sua complexidade computacional $O(n \log n)$, assim como o uso da pesquisa binária com $O(\log n)$, sendo *n* a quantidade de elementos do vetor código,

ou seja, a dimensão *K* do Algoritmo Competitivo ($n = K$). Para melhorar o desempenho do AC com ENNS, no momento de calcular a média, temos que o denominador da média dos vetores código será sempre o mesmo, *K* (dimensão do dicionário), logo, em vez de realizarem várias divisões (maior custo computacional, veja Tabela V) essa é substituída por uma única divisão e sucessivas multiplicações (isto será explicado mais adiante).

Nas simulações foi usada a imagem "Lena" (Figura 3), com dimensões de 256x256, 8 bits por pixel. Os seguintes parâmetros foram usados nas simulações: $K = 16$ e *N* assumindo os valores 32, 64, 128, 256 e 512. A implementação dos algoritmos, escritos em linguagem C, foi feita no Visual Studio 2012. Os experimentos foram realizados num *Intel Core i5-3210M (2,50 GHz) DELL Notebook Vostro 3460*.

Figura 3. Imagem "Lena"com resolução 256x256, 8bpp.

VI. Resultados

Os resultados apresentados nas Tabelas II, III e IV são médias calculadas a partir de 20 execuções. Além disso, foram usados dicionários iniciais iguais para cada valor de *N* em cada algoritmo. Nas Tabelas II, III e IV são mostrados o tempo de execução e o número de operações aritméticas de cada versão implementada.

Tabela II
Tempo de execução em segundos e número de operações aritméticas realizadas pelo AC simples.

N	Tempo	Adição	Subtração	Multiplicação	Divisão	Comparação
32	3,125	1E+07	1E+07	1E+07	1	7E+05
64	6,306	3E+07	3E+07	3E+07	1	1E+06
128	12,513	5E+07	5E+07	5E+07	1	3E+06
256	24,988	1E+08	1E+08	1E+08	1	6E+06
512	49,953	2E+08	2E+08	2E+08	1	1E+07

Tabela III
Tempo de execução em segundos e número de operações aritméticas realizadas pelo AC com PDS.

N	Tempo	Adição	Subtração	Multiplicação	Divisão	Comparação
32	0,976	1E+06	1E+06	1E+06	1	2E+06
64	1,788	2E+06	2E+06	2E+06	1	5E+06
128	3,022	4E+06	4E+06	4E+06	1	9E+06
256	5,154	6E+06	6E+06	6E+06	1	2E+07
512	8,640	1E+07	1E+07	1E+07	1	3E+07

Tabela IV
Tempo de execução em segundos e número de operações aritméticas realizadas pelo AC com ENNS.

N	Tempo	Adição	Subtração	Multiplicação	Divisão	Comparação
32	0,280	2E+06	2E+05	2E+05	2	4E+05
64	0,407	5E+06	4E+05	4E+05	2	6E+05
128	0,697	9E+06	6E+05	6E+06	2	1E+06
256	1,200	2E+07	1E+06	1E+06	2	2E+06
512	2,118	4E+07	2E+06	2E+06	2	4E+06

De acordo com as tabelas, o AC com PDS e com ENNS reduzem os números de adições, subtrações e multiplicações, em relação ao AC simples, da mesma forma que o tempo de execução foi melhorado. Há também a redução de subtrações, multiplicações e comparações do AC com ENNS em relação ao AC com PDS. A redução do número de adições, subtrações e multiplicações no AC com PDS é devido a uma possível saída antecipada no momento de calcular a distância euclidiana quadrática. No geral, a redução no número de subtrações e multiplicações é maior em comparação às adições. Isso se deve ao cálculo realizado para encontrar o vencedor (distância euclidiana quadrática) que realiza sempre uma operação de adição a menos em relação as operações de subtração e multiplicação, essa mesma justificativa é observada no AC com ENNS, já que o mesmo também utiliza o PDS. No entanto, são observadas outras variações nos números de operações do AC com ENNS, em relação ao AC com PDS. A redução do número de multiplicações e subtrações é justificada pela forma como o algoritmo ENNS funciona, ou seja, utilizando a ordenação das médias dos vetores código para restringir a busca entre uma faixa de médias mais semelhantes a um dado vetor de treino. Outra variação na tabela são os números de divisões e adições. O número de adições aumenta por causa do cálculo da média dos vetores código, assim como a divisão. Como as divisões do cálculo da média são sempre pelo tamanho dos vetores, *K*, então essas divisões são constantes. Logo, esse cálculo é realizado uma única vez, como

$$Constante = \frac{1}{K}.$$

Para as demais médias é multiplicado o resultado dessa divisão (*Constante*) pela soma dos componentes de cada vetor. Na Tabela V é possível perceber que a divisão é a operação que leva mais tempo para ser executada, portanto, essa modificação gera uma economia no tempo do projeto do dicionário. Os valores de comparação do AC com ENNS são devido ao uso do algoritmo Heapsort para ordenar os vetores após o cálculo da média, da mesma forma que é necessário o uso da pesquisa binária para fazer a busca do vetor código vencedor.

Seja *M* o número de vetores de treino, ou seja, a quantidade de blocos de tamanho *K* que compõe a imagem, os números de operações aritméticas obtidas no Algoritmo Competitivo são definidos em [2], como:

- Multiplicação: $[1 + (1 + N)KM]\eta_{AC} - 1$;
- Subtração: $[1 + (1 + N)KM]\eta_{AC} + 1$;
- Adição: $[1 + (K - 1)NM + KM]\eta_{AC} - 1$;

Tabela V
TEMPO PARA REALIZAR 1 BILHÃO DE REPETIÇÕES DA MESMA OPERAÇÃO.

Operação	Tempo em Segundos
Adição	2,589
Subtração	2,496
Multiplicação	2,980
Divisão	9,516
Comparação	2,402

- Divisão: 1;
- Comparação: $(N-1)M\eta_{AC}$.

Para o Algoritmo Competitivo com PDS os valores não são constantes, pois o número de operações aritméticas varia de acordo com o dicionário inicial. Mas observa-se na Tabela VI que o número de distâncias euclidianas calculadas em média são 75,66% menores no AC com PDS, em relação ao AC com busca total, o que leva à redução quantidade de operações aritméticas, que por sua vez diminui o tempo de execução. Ainda na Tabela VI vemos a grande redução de distâncias euclidianas calculadas em comparação ao AC com busca total, por causa do uso do ENNS, a partir da segunda iteração, que compara as médias e não a distância eucliana (a distância eucliana só é calculada na primeira iteração).

Tabela VI
NÚMERO DE DISTÂNCIAS EUCLIDIANAS CALCULADAS PELO ALGORITMO COMPETITIVO EM RELAÇÃO AO AC COM PDS. PORCENTAGEM DE REDUÇÃO ENTRE PARÊNTESES EM RELAÇÃO AO AC.

N	AC	PDS	ENNS
32	5E+07	1E+07 (70,80%)	6E+04 (99,87%)
64	1E+08	2E+07 (77,25%)	9E+04 (99,90%)
128	2E+08	3E+07 (80,85%)	1E+05 (99,94%)
256	4E+08	7E+07 (82,47%)	1E+05 (99,97%)
512	8E+08	1E+08 (86,75%)	2E+05 (99,98%)

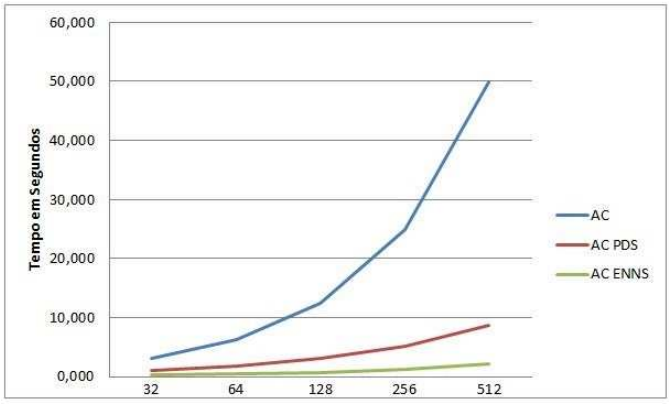

Figura 4. Variação do tempo de projeto de quantizadores em função do tamanho do dicionário (N).

A Figura 4 mostra que o Algoritmo competitivo, no geral, necessita de mais tempo para projetar um dicionário. As outras duas variações de AC apresentam algumas relações. Para dicionários de tamanho até $N = 64$ o AC com PDS se mostra um pouco menos eficiente do que o AC com ENNS. À medida que o tamanho do dicionário aumenta as curvas do AC com PDS e AC com ENNS se afastam ainda mais. Para $N = 512$ existe uma diferença de 41,313 segundos entre o AC e o AC com PDS, o que representa uma redução de 82,70% do tempo gasto e 95,76% do tempo de execução do AC original. As reduções de tempo são mostradas nas Tabelas VII, VIII, IX e X, onde no geral o melhor desempenho é do Algoritmo Competitivo com ENNS seguido do AC com PDS e por último o AC simples com busca total.

Tabela VII
TEMPO DE EXECUÇÃO EM SEGUNDOS GASTO PELO ALGORITMO COMPETITIVO NATURAL E AC COM PDS. PORCENTAGEM DE REDUÇÃO ENTRE PARÊNTESES.

N	AC	PDS
32	3,125	0,976 (68,76%)
64	6,306	1,788 (71,64%)
128	12,513	3,022 (75,84%)
256	24,988	5,154 (79,37%)
512	49,953	8,640 (82,70%)

Tabela VIII
TEMPO DE EXECUÇÃO EM SEGUNDOS GASTO PELO ALGORITMO COMPETITIVO NATURAL E AC COM ENNS. PORCENTAGEM DE REDUÇÃO ENTRE PARÊNTESES.

N	AC	ENNS
32	3,125	0,280 (91,04%)
64	6,306	0,407 (93,54%)
128	12,513	0,697 (94,42%)
256	24,988	1,200 (94,43%)
512	49,953	2,118 (95,76%)

Tabela IX
TEMPO DE EXECUÇÃO EM SEGUNDOS GASTO PELO AC COM PDS E AC COM ENNS. PORCENTAGEM DE REDUÇÃO ENTRE PARÊNTESES.

N	PDS	ENNS
32	0,976	0,280 (71,31%)
64	1,788	0,407 (77,23%)
128	3,022	0,697 (76,93%)
256	5,154	1,200 (76,71%)
512	8,640	2,118 (75,48%)

VII. CONSIDERAÇÕES FINAIS

Algoritmos de aprendizagem não-supervisionada de redes neurais têm um amplo aspecto de aplicações, dentre as quais reconhecimento de padrões, minerações de dados e compressão de sinais. Em se tratando de compressão de voz e imagem, por exemplo, algoritmos de aprendizagem competitiva podem ser usados para projeto de dicionários aplicados à compressão baseada em quantização vetorial (QV). Neste trabalho, foi apresentada uma alternativa de aceleração de um algoritmo competitivo aplicado ao projeto de dicionários para QV de imagens. A aceleração consiste em acomodar o algoritmo ENNS (*Equal-average Nearest Neighbor Search*) na etapa de determinação do neurônio vencedor. Resultados de simulações envolvendo projeto de dicionários para QV de

imagens monocromáticas revelam que a alternativa apresentada leva a economia de até cerca de 93,84% no tempo de projeto dos dicionários.

AGRADECIMENTOS

Este artigo contou com o apoio financeiro do CNPq, do programa PIBIC da Universidade Católica de Pernambuco.

REFERÊNCIAS

[1] E. A. LIMA, P. H. ESPÍRITO SANTO and F. MADEIRO,"Sequências de Baixa Discrepância Aplicadas à Inicialização do Algoritmo Linde-Buzo-Gray", TEMA Tend. Mat. Apl. Comput., Vol. 11, n. 3, pp. 217-229, 2010.

[2] F. MADEIRO, W. T. A. LOPES, B. G. AGUAR NETO and M. S. ALENCAR,"Complexidade Computacional de um Algoritmo Competitivo Aplicado ao Projeto de Quantizadores Vetoriais", Learning and Nonlinear Models - Revista da Sociedade Brasileira de Redes Neurais (SBRN), Vol. 2, n. 1, pp. 34-48, 2004.

[3] P. H. E. SANTO, R. C. ALBUQUERQUE, D. C. CUNHA and F. MADEIRO, "On Frequency Sensitive Competitive Learning for VQ Codebook Design", in 10th Brazilian Symposium on Neural Networks, pp. 135-140, 26-30, October 2008.

[4] C. ZHU and L. M. PO,"Partial Distortion Sensitive Competitive Learning Algorithm for Optimal Codebook Design", in IEE Electronics, Vol. 32, n. 19, pp. 1757-1758, 12th September 1996.

[5] A. GERSHO and R. M. GRAY, "Vector Quantization and Signal Compression", Kluwer Academic Publishers, Boston, MA, 1992.

[6] R. M. GRAY, "Vector Quantization". *IEEE ASSP Magazine*, pp. 4-29, April 1984.

[7] C. D. BEI and R. M. GRAY, "An Improvement of the Minimum Distortion Enconding Algorithm for Vector Quantization", in *IEEE Transactions on Communications*, Vol. COM-33, n. 10, pp. 1132-1133, October 1985.

[8] S. C. CHU, Z. M. LU and J. S. PAN, "Hadamard Transform Based Fast Codeword Search Algorithm for High-dimensional VQ Enconding", in Information Sciences, Vol. 177, pp. 734-746, February 2007.

[9] T. H. CORMEN, C. E. LEISERSON, R. L. RIVEST and C. STEIN, Algoritmos Teoria e Prática, Elsevier, 2002.

[10] L. GUAN and M. KAMEL, "Equal-average Hyperplane Partitioning Method for Vector Quantization of Image Data", Pattern Recognition, Letters, Vol. 13, n. 10, pp. 693-699, 1992.

[11] A. K. KRISHNAMURTHY, S. C. AHALT and D.E. MELTON, "Neural Networks for Vector Quantization of Speech and Images", IEEE J Sel. Areas Commum., SAC-8, (8), pp. 1449-1457, 1990.

[12] F. MADEIRO, R. R. A. GALVÃO, F. A. B. S. SILVEIRA and C. D. CUNHA, "Uma Alternativa de Aceleração do Algoritmo Fuzzy K-Means Aplicado à Quantização Vetorial", TEMA Tendências em Matemática Aplicada e Computacional, Vol. 13, pp. 193-206, 2012.

[13] R. R. A. GALVÃO, F. MADEIRO, D. C. CUNHA and P. H. ESPÍRITO SANTO, "Aceleração e Avaliação da Complexidade Computacional do Algoritmo Competitivo Sensível a Distância Parcial Aplicado ao Projeto de Quantizadores Vetorial". In: Congresso Brasileiro de Automática, 2012, Campina Grande, PB. Anais do Congresso Brasileiro de Automática, 2012.

[14] C. R. B. AZEVEDO, R. A. AZEVEDO, E. L. BISPO JUNIOR, T. A. E. FERREIRA, W. T. A. LOPES and F. MADEIRO, "Um Algoritmo Memético para a Otimização de Quantizadores Vetoriais", Learning and Nonlinear Models, Vol. 5, pp. 1-15, 2008.

[15] C. R. B. AZEVEDO, F. E. A. G. AZEVEDO, W. T. A. LOPES and F. MADEIRO, "Terrain-Based Memetic Algorithms for Vector Quantizer Design. In: International Workshop on Nature Inspired Cooperative Strategies for Optimization", Proceedings of the International Workshop on Nature Inspired Cooperative Strategies for Optimization (NICSO), Puerto de La Cruz, Tenerife, 2008.

[16] N. B. KARAYIANNIS and P.-I. PAI and N. ZERVOS, "Image Compression Based on Fuzzy Algorithms for Learning Vector Quantization and Wavelet Image Decomposition", IEEE Transactions on Image Processing, Vol. 7, n. 8, pp. 1223–1230, August 1998.

[17] T. KOHONEN, "The Self-Organizing Map". Proceedings of the IEEE, Vol. 78, n. 9, pp. 1464–1480, September 1990.

[18] H. B. KEKRE, T. K. SARODE and J. K. SAVE, "Error Vector Rotation Using Kerkre Transform for Efficient Clustering in Vector Quantization", in International Journal of Advances in Engineering & Technology, Vol. 4, Issue 1, pp. 256-264, July 2012.