Classificação de grandes bases de dados utilizando algoritmo de Máquina de Aprendizado Extremo

Carlos Alexandre Siqueira da Silva
Universidade Federal do Espírito Santo - UFES
Av. Fernando Ferrari, 514, Goiabeiras - Vitória - ES - CEP 29075-910
carlosalexandress@gmail.com

Renato A. Krohling
Universidade Federal do Espírito Santo - UFES
Av. Fernando Ferrari, 514, Goiabeiras - Vitória - ES - CEP 29075-910
krohling.renato@gmail.com

RESUMO

Máquinas de Aprendizado Extremo (*Extreme Learning Machines*, ou ELM) são redes neurais com reduzido tempo de treinamento e grande poder de generalização. Devido a sua forma analítica de obtenção dos valores de saída da rede, e a consequente ausência de ajuste dos pesos, o ELM necessita de um número maior de neurônios, o que aumenta o seu custo computacional quando utilizado para classificação de grandes bases. Uma das maneiras de minimizar esse problema é utilizar arquiteturas paralelas para as operações do algoritmo que exigem maior poder computacional. No presente trabalho, são descritas duas variações do ELM implementadas numa arquitetura paralela utilizando a GPU (*Graphic Processing Unit*), demonstrando sua performance na classificação de bases grandes e comparando com outras implementações da literatura.

PALAVRAS CHAVE. ELM, classificação, grandes bases.

Tópicos: Metaheuristicas

ABSTRACT

Extreme Learning Machine (ELM) are neural networks with reduced training time and great generalization capability. Due to its analytic form for obtaining the network output values, and the consequent absence of setting of weights, ELM requires a larger number of neurons, which increases the computational cost when used for large databases classification. One way to minimize this problem is to use parallel architectures for algorithm operations that requires more computational power. In this paper we describe two ELM variations implemented in a parallel architecture using the Graphic Processing Unit (GPU), demonstrating its performance in large databases classification and comparing with other literature implementations.

KEYWORDS. ELM, classification, big data.

Paper topics: Metaheuristics

1. Introdução

O algoritmo de treinamento de redes neurais Máquina de Aprendizado Extremo (do inglês *Extreme Learning Machine*, ou ELM) [Huang et al., 2004], [Huang et al., 2006] se consolidou como uma opção competitiva em redes neurais devido ao seu reduzido tempo de treinamento e grande poder de generalização, contrapondo-se aos algoritmos baseados no gradiente do erro como o *backpropagation*.

O ELM não necessita de ajuste dos pesos, o que o torna mais rápido do que os algoritmos iterativos. Porém, para que o modelo apresente boa capacidade de generalização, é necessário um número maior de neurônios na camada escondida do que em uma rede neural *feedforward* convencional. Isso faz com que a etapa de treinamento da rede se torne computacionalmente dispendiosa, impossibilitando que o algoritmo ELM seja utilizado em grandes bases de dados.

Algumas variações do ELM já foram propostas visando minimizar o seu custo computacional. [Xin et al., 2014] propuseram uma implementação do ELM baseada em MapReduce, de forma a utilizar o paralelismo do ambiente MapReduce para calcular a inversa generalizada de Moore-Penrose, enquanto [He et al., 2013] propuseram uma implementação que utiliza o MapReduce para paralelizar as demais operações matriciais do algoritmo.

[Zhou et al., 2015] apresentaram uma variação do ELM que divide a rede em camadas, de forma a dividir o custo computacional utilizando um número reduzido de neurônios em cada camada. Já [Akusok et al., 2015] desenvolveram um framework para criação de redes neurais ELM utilizando GPUs (do inglês *Graphics Processing Unit*).

No presente trabalho, serão apresentadas duas variações do algoritmo ELM, implementadas em uma arquitetura paralela utilizando a GPU para classificação de grandes bases de dados. O restante deste artigo está organizado da seguinte forma: Na seção 2 são descritos os algoritmos utilizados; a seção 3 descreve os experimentos computacionais realizados; os resultados e discussões encontram-se na seção 4; e a seção 5 traz a conclusão.

2. Descrição dos Algoritmos

2.1. Algoritmo ELMP

O algoritmo ELM foi proposto por [Huang et al., 2004] e [Huang et al., 2006], tendo como base redes neurais do tipo *feedforward* com uma única camada escondida (do inglês *Single-Layer Feed-forward Networks*, ou SLFN). De acordo com os autores, os parâmetros da camada escondida de uma rede neural (pesos dos neurônios e valores de bias) não precisam ser ajustados, podendo ser definidos de forma aleatória (ou semi-aleatória) e independente dos dados de entrada. Os pesos da camada de saída são obtidos de forma analítica, através do método dos mínimos quadrados. A Figura 1 apresenta um modelo de rede ELM.

Formalização do método ELM: Considere um conjunto de treinamento (x_i, t_i), $x_i \in R^d$, $t_i \in R^m$, i = 1, ..., N, uma função de ativação da camada escondida G(w, b, x), e um número L de nós na camada escondida. A relação entre os dados x_i de entrada e os valores t_i de saída da rede é dada por:

$$t_i = \sum_{j=1}^{L} \beta_i G(w_j, b_j, x_i), i = 1, ..., N, \quad (1)$$

sendo β_i os pesos da camada de saída da rede, w_j e b_j os pesos e os bias da camada escondida da rede. A equação (1) pode ser escrita na forma matricial Hβ = T, agrupando-se as saídas dos neurônios da camada escondida em uma matriz H, conforme equações (2) e (3).

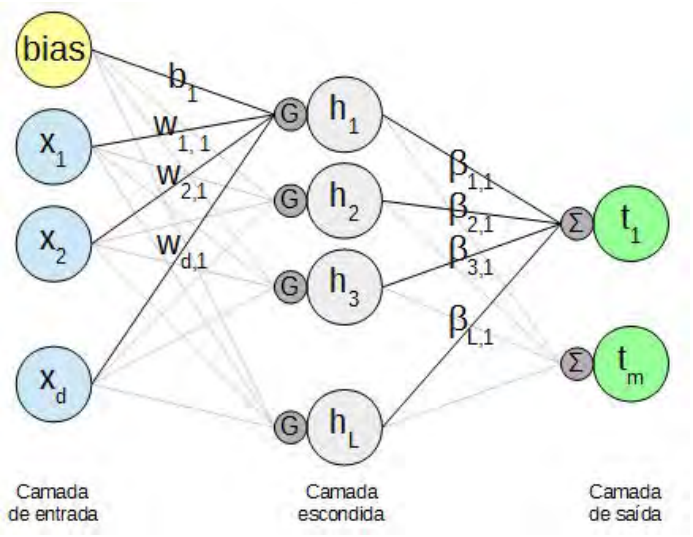

Figura 1: Modelo de rede ELM

$$H = \begin{bmatrix} G(w_1, b_1, x_1) & \dots & G(w_L, b_L, x_1) \\ \vdots & \ddots & \vdots \\ G(w_1, b_1, x_N) & \dots & G(w_L, b_L, x_N) \end{bmatrix} \tag{2}$$

$$\beta = \begin{bmatrix} \beta_1^T \\ \vdots \\ \beta_L^T \end{bmatrix}, T = \begin{bmatrix} t_1^T \\ \vdots \\ t_N^T \end{bmatrix} \tag{3}$$

O sistema linear Hβ = T pode ser resolvido empregando-se a inversa generalizada (pseudo-inversa) de Moore-Penrose da matriz H, representada por $H^{\dagger}$. Existem diferentes maneiras de se calcular a pseudo-inversa de uma matriz, entre elas o método da projeção ortogonal, método iterativo e a decomposição em valores singulares (do inglês *Singular Value Decomposition*, ou SVD) [Ben-Israel e Greville, 2003].

Segundo a teoria da regressão *ridge* [Hoerl e Kennard, 1970], a adição de um fator de regularização α pequeno na diagonal de uma matriz garante a não-singularidade dessa matriz, possibilitando que o método da projeção ortogonal seja utilizado no cálculo geral da pseudo-inversa. Conforme demonstrado por [Huang et al., 2012], esse fator de regularização torna a solução obtida mais estável e com maior poder de generalização. Dessa forma, para um dado α pequeno, o vetor β pode ser obtido pelo método da projeção ortogonal através da fórmula

$$\beta = H^{\dagger}T \tag{4}$$

$$H^{\dagger} = (H^T H + \alpha I)^{-1} H^T \tag{5}$$

O método ELM pode ser descrito conforme o Algoritmo 1.

Algoritmo 1: ELM

Entrada: Conjunto de treinamento (x_i, t_i), $x_i \in R^d$, $t_i \in R^m$, i = 1, ..., N; Função de ativação da camada escondida G(w, b, x); Número L de nós na camada escondida

Saída: Pesos β da camada de saída

1 **início**
2 Atribua valores aleatórios (w_i, b_i), i = 1, ..., L, aos nós da camada escondida.
3 Calcule a matriz de saída da camada escondida H.
4 Calcule os pesos β da camada de saída.
5 **fim**

No presente trabalho, foi implementada uma versão paralela do algoritmo ELM, aqui denominada ELMP. A principal diferença do ELMP para o ELM é que as multiplicações de matrizes presentes nos passos 3 e 4 do Algoritmo 1 são executadas na GPU, reduzindo o tempo computacional do algoritmo sem causar perda de acurácia. Devido às dimensões das bases utilizadas, foi necessário realizar o "fatiamento"(do inglês *slicing*) das matrizes em blocos menores antes de processá-las na GPU, evitando estouro de memória. A abordagem paralela utilizada neste trabalho é próxima à descrita por [Akusok et al., 2015].

2.2. Algoritmo ELMAEP

Em [Zhou et al., 2015], é apresentada uma variação do algoritmo ELM que utiliza um *autoencoder* para inicialização dos pesos da camada escondida da rede.

Autoencoders são redes neurais que codificam as entradas de uma rede em um modelo que permita a reconstrução dessas entradas. Isso significa que os valores de saídas esperados para a rede são os próprios valores de entrada.

Estudos envolvendo *autoencoders* em arquiteturas de *deep learning* [Hinton et al., 2006], [Huang et al., 2016] apontam que essa é uma abordagem promissora para a extração de características de uma base, para a redução de sua dimensionalidade e para o pré-treinamento dos dados.

O algoritmo proposto por [Zhou et al., 2015] (aqui referenciado como ELMAE) utiliza uma etapa de *autoencoding* da base de treinamento através de uma rede ELM, de forma que os pesos de saída β_i obtidos equivalem à matriz de reconstrução do *autoencoder*. O algoritmo ELMAE utiliza a matriz β^T, transposta de β, como matriz de pesos iniciais da camada escondida de uma outra rede ELM, obtendo acurácia superior à inicialização aleatória dos pesos. A Figura 2 apresenta um modelo do algoritmo ELMAE.

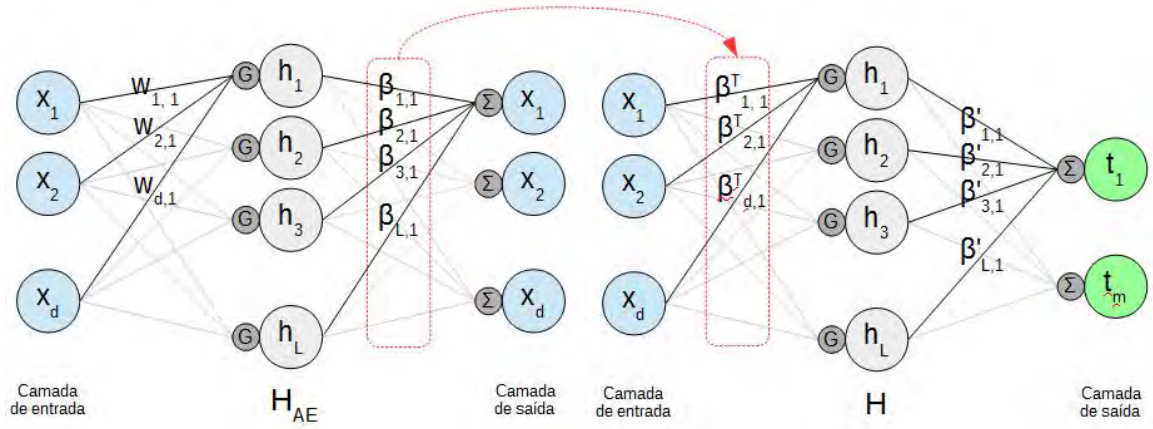

Figura 2: Modelo de rede ELM Autoencoder

No presente trabalho, foi implementada uma versão paralela do algoritmo ELMAE, aqui denominada ELMAEP. Assim como o algoritmo ELMP, o algoritmo ELMAEP também realiza as

multiplicações de matrizes necessárias ao treinamento da rede na GPU, reduzindo o tempo computacional do algoritmo sem causar perda de acurácia.

3. Experimentos Computacionais

3.1. Bases de dados utilizadas

Para esse trabalho foram utilizadas 7 bases de dados de domínio público disponíveis na *UCI Machine Learning Repository* [UCI, 2016]. As bases são apresentadas na Tabela 1 com a quantidade de atributos, número de classes e número de instâncias. Foram consideradas tanto bases com grande número de entradas (dimensões variando de 18 a 784 atributos) quanto bases com grande quantidade de instâncias.

Base	# Atributos	# Classes	# Instâncias Treinamento	# Instâncias Teste
DNA	180	3	2000	1186
Isolet	617	26	6238	1559
Mnist	784	10	60000	10000
Covertype	54	7	406709	174303
Susy	18	2	3500000	1500000
Hepmass	18	2	7000000	3500000
Higgs	28	2	7700000	3300000

Tabela 1: Bases de dados utilizadas

Todas as bases foram normalizadas no intervalo [0, 1]. Para as bases Mnist, Hepmass, Isolet e DNA, foram respeitadas as partições de treinamento e teste estabelecidas na literatura. As bases Susy, Higgs e Covertype foram particionadas na proporção 70%-30% (treinamento - testes).

3.2. Parâmetros utilizados

Foram realizados testes com sete configurações de neurônios da camada escondida (50, 100, 200, 300, 400, 500 e 600 neurônios), para os dois algoritmos testados. A acurácia apresentada refere-se à média da taxa de acerto para dez execuções do algoritmo. Foi utilizada a função sigmóide como função de ativação da camada escondida da rede. Como fator de regularização para o cálculo da inversa generalizada de Moore-Penrose, foi utilizado o valor de $5x10^{-8}$.

Para a execução do algoritmo ELMAEP, foi utilizada a partição de treinamento para o *autoencoding* inicial.

Os dois algoritmos foram executados em um ambiente híbrido GPU + CPU, sendo direcionadas à GPU as operações de maior custo computacional (multiplicação e inversão das matrizes). Foi utilizado para os testes um computador Intel com processador Core i7 com 3.60GHz e 16GB de memória RAM, equipado com uma placa de vídeo NVidia GeForce GTX 745. Para maior portabilidade dos algoritmos, os protótipos foram implementados na linguagem OpenCL [OpenCL, 2016], compatível com placas de vídeo NVidia, AMD e Intel.

4. Resultados e Discussões

A Tabela 2 apresenta o comparativo da acurácia média e o tempo de treinamento entre os algoritmos executados e os encontrados na literatura. Foram consideradas para comparação as execuções do ELMP e ELMAEP com 600 neurônios na camada escondida, por apresentarem maior acurácia.

Base	Algoritmo	Acuracia (%)	Tempo Treino (s)	# Neuronios
DNA	ELMP	88.85 ± 0.50	1	600
	ELMAEP	90.89 ± 0.79	3	600
	LS-IELM [Guo et al., 2014]	$\mathbf{95.24 \pm 0.64}$	-	200
Isolet	ELMP	90.71 ± 0.65	5	600
	ELMAEP	94.40 ± 0.28	11	600
Mnist	ELMP	91.82 ± 0.19	28	600
	ELMAEP	92.60 ± 0.15	61	600
	AE-S-ELM [Zhou et al., 2015]	$\mathbf{98.89 \pm 0.06}$	4347	500
	HP-ELM [Akusok et al., 2015]	≈ 95	≈ 35	4000
Covertype	ELMP	$\mathbf{79.53 \pm 0.10}$	63	600
	ELMAEP	78.91 ± 0.10	129	600
	MRAC+ [Bechini et al., 2016]	78.09 ± 0.16	504	-
Susy	ELMP	$\mathbf{79.62 \pm 0.04}$	514	600
	ELMAEP	79.32 ± 0.05	1045	600
	MRAC+ [Bechini et al., 2016]	78.22 ± 0.04	738	-
Higgs	ELMP	64.95 ± 0.05	1176	600
	ELMAEP	65.10 ± 0.05	2476	600
	MRAC+ [Bechini et al., 2016]	65.90 ± 0.05	6141	-
	HP-ELM [Akusok et al., 2015]	$\approx \mathbf{70}$	≈ 3500	4000
Hepmass	ELMP	84.43 ± 0.09	1059	600
	ELMAEP	84.56 ± 0.04	2221	600

Tabela 2: Acurácia, tempo de treinamento e número de neurônios

O algoritmo HP-ELM [Akusok et al., 2015] não apresentou uma tabela de tempo de treinamento e acurácia para as bases Mnist e Higgs, apenas um gráfico. Portanto, os valores na Tabela 2 são aproximados. Foi utilizada para comparação a configuração de 4000 neurônios para esse algoritmo por ser a maior com dados de tempo de execução no artigo.

Os algoritmos ELMP e ELMAEP se mostraram mais rápidos do que os demais nas bases Mnist, Covertype e Higgs. O ELMP ainda se mostrou mais rápido para a base Susy. A acurácia foi superior nas bases Covertype e Susy.

Pode-se observar que nas bases com maior número de instâncias (Covertype, Susy, Hepmass e Higgs), o algoritmo ELMAEP apresenta pouca ou nenhuma melhoria da acurácia em relação ao ELMP.

A Figura 3 apresenta a acurácia obtida para as bases DNA e Isolet para cada configuração de neurônios da camada escondida. Para as duas bases observa-se que a inicialização dos pesos da camada escondida através de um *autoencoder* aumenta a acurácia da rede e torna o modelo mais robusto, pois há redução do desvio padrão dos resultados.

O aumento do número de neurônios acima de 400 na base DNA provoca o sobre-ajuste (do inglês *overfitting*) do modelo, ocasionando queda na acurácia. Já a base Isolet começa a estabilizar ao se aproximar dos 600 neurônios, indicando que o aumento desse parâmetro não terá impacto significativo na acurácia.

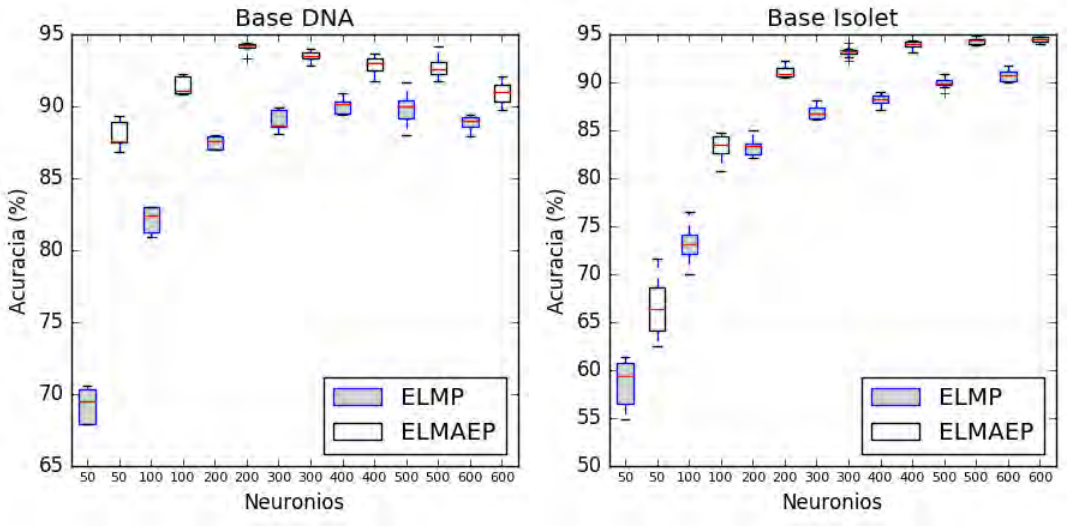

Figura 3: Bases de dados DNA e Isolet

A Figura 4 apresenta a acurácia obtida para as bases Mnist e Covertype para cada configuração de neurônios da camada escondida. Observa-se para a base Mnist que a inicialização dos pesos da camada escondida através de um *autoencoder* aumenta a acurácia da rede. Não há indícios de que a robustez do modelo tenha sido melhorada. Já para a base Covertype, a inicialização dos pesos da camada escondida através do *autoencoder* não trouxe melhora para a acurácia ou para a robustez do modelo.

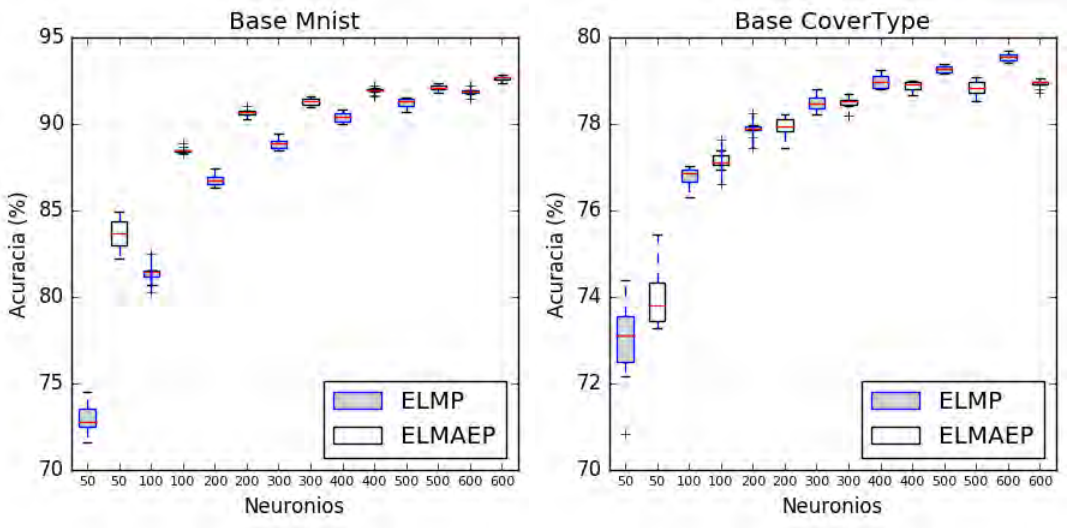

Figura 4: Bases de dados Mnist e Covertype

A Figura 5 apresenta a acurácia obtida para as bases Susy e Hepmass para cada configuração de neurônios da camada escondida. Para a base Susy, o algoritmo ELMAEP apresentou acurácia inferior ao ELMP. Para a base Hepmass, houve uma pequena melhora na acurácia ao se utilizar o algoritmo ELMAEP.

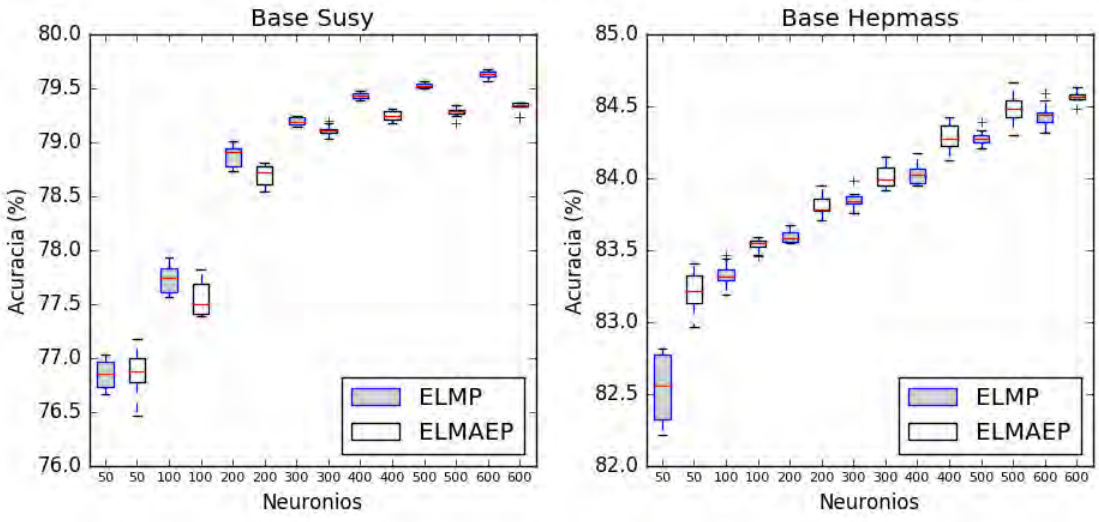

Figura 5: Bases de dados Susy e Hepmass

A Figura 6 apresenta a acurácia obtida para a base Higgs para cada configuração de neurônios da camada escondida. Observa-se para essa base que o algoritmo ELMAEP apresenta bom desempenho para até 200 neurônios. Para configurações acima desse valor, há pouca melhora na acurácia em relação ao algoritmo ELMP, sendo que ambos ficam próximos de uma acurácia de 65%.

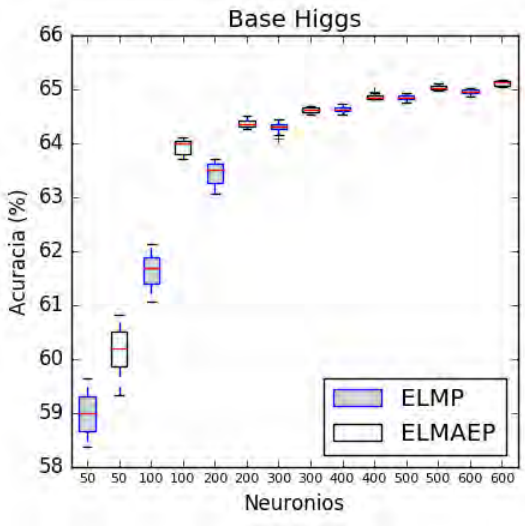

Figura 6: Base de dados Higgs

A Figura 7 e a Figura 8 apresentam os tempos de treinamento dos algoritmos ELMP e ELMAEP, respectivamente. As sete bases de dados foram separadas em dois grupos devido à ordem de grandeza da escala de tempo, para facilitar a visualização.

O algoritmo ELMAEP possui uma etapa de *autoencoding* na sua fase de treinamento, executado por meio de outra rede neural do tipo ELM. Dessa forma, o tempo de treinamento desse algoritmo aproxima-se do dobro do tempo de treinamento do algoritmo ELMP.

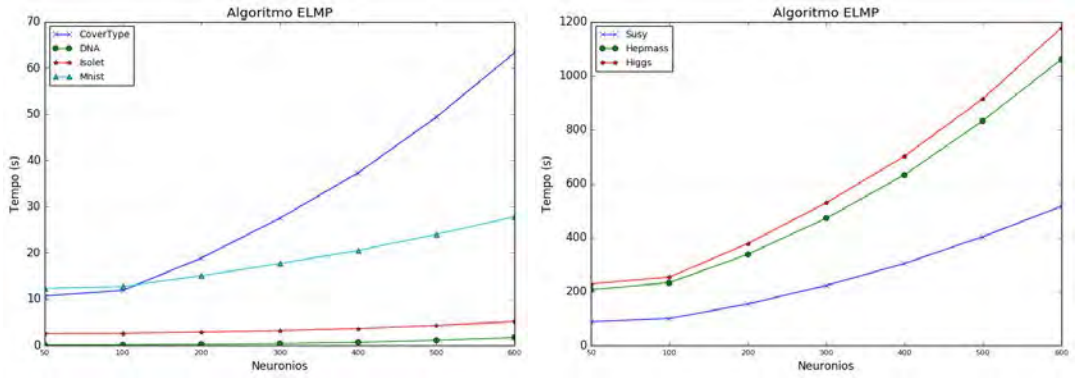

Figura 7: Tempo de treinamento do algoritmo ELMP

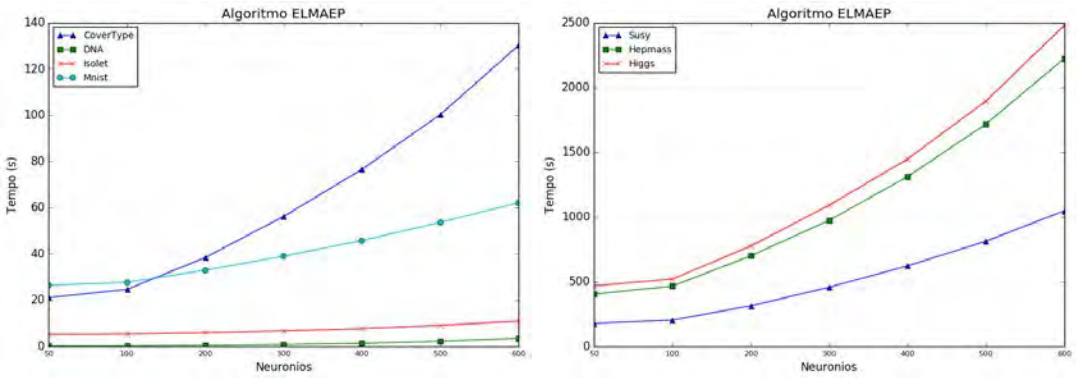

Figura 8: Tempo de treinamento do algoritmo ELMAEP

5. Conclusão

No presente trabalho foi apresentado um comparativo entre duas variações do algoritmo ELM (ELMP e ELMAEP) para bases grandes em uma implementação paralela utilizando recursos da GPU.

A arquitetura paralela da GPU mostrou-se muito promissora para o aumento da performance dos algoritmos, o que pode ser comprovado pelo tempo de treinamento menor dos algoritmos apresentados em quase todas as bases.

A adoção de um *autoencoder* para inicialização dos pesos da camada escondida da rede ELM trouxe melhora na acurácia em 5 das 7 bases, mas aumentou consideravelmente o tempo de treinamento da rede em relação à inicialização aleatória. A adoção de um método de inicialização semi-aleatório poderá ser mais vantajoso, e será fruto de estudos futuros.

Referências

Akusok, A., Bjork, K.-M., Miche, Y., e Lendasse, A. (2015). High-performance extreme learning machines: a complete toolbox for big data applications. *Access, IEEE*, 3:1011–1025.

Bechini, A., Marcelloni, F., e Segatori, A. (2016). A MapReduce solution for associative classification of big data. *Information Sciences*, 332:33–55.

Ben-Israel, A. e Greville, T. N. (2003). *Generalized inverses: theory and applications*, volume 15. Springer Science & Business Media.

Guo, L., Hao, J.-h., e Liu, M. (2014). An incremental extreme learning machine for online sequential learning problems. *Neurocomputing*, 128:50–58.

He, Q., Shang, T., Zhuang, F., e Shi, Z. (2013). Parallel extreme learning machine for regression based on MapReduce. *Neurocomputing*, 102:52–58.

Hinton, G. E., Osindero, S., e Teh, Y.-W. (2006). A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554.

Hoerl, A. E. e Kennard, R. W. (1970). Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(1):55–67.

Huang, G.-B., Zhou, H., Ding, X., e Zhang, R. (2012). Extreme learning machine for regression and multiclass classification. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 42(2):513–529.

Huang, G.-B., Zhu, Q.-Y., e Siew, C.-K. (2004). Extreme learning machine: a new learning scheme of feedforward neural networks. In *2004 IEEE International Joint Conference on Neural Networks*, volume 2, p. 985–990.

Huang, G.-B., Zhu, Q.-Y., e Siew, C.-K. (2006). Extreme learning machine: theory and applications. *Neurocomputing*, 70(1):489–501.

Huang, W.-b., Sun, F.-c., et al. (2016). Building feature space of extreme learning machine with sparse denoising stacked-autoencoder. *Neurocomputing*, 174:60–71.

OpenCL (2016). OpenCL. `https://www.khronos.org/opencl/`. Acessado: 2016-04-01.

UCI (2016). UCI machine learning repository. `http://archive.ics.uci.edu/ml`. Acessado: 2016-04-01.

Xin, J., Wang, Z., Chen, C., Ding, L., Wang, G., e Zhao, Y. (2014). ELM*: distributed extreme learning machine with MapReduce. *World Wide Web*, 17(5):1189–1204.

Zhou, H., Huang, G.-B., Lin, Z., Wang, H., e Soh, Y. C. (2015). Stacked extreme learning machines. *IEEE Transactions on Cybernetics*, 45(9):2013–2025.