Revista Brasileira de Computação Aplicada, Julho, 2019

DOI: 10.5335/rbca.v11i2.9089
Vol. 11, Nº 2, pp. 13–21
Homepage: seer.upf.br/index.php/rbca/index

ISSN 2176-6649

ARTIGO ORIGINAL

Implementações paralelas para o algoritmo Online Sequential Extreme Learning Machine aplicado à previsão de material particulado

Luís Fernando Lopes Grim[1, 2], Jorge Andrés Bueno Barajas[1] and André Leon Sampaio Gradvohl[1]

[1]Universidade Estadual de Campinas and [2]Instituto Federal de São Paulo

*lgrim@ifsp.edu.br; andresbueno@gmail.com; gradvohl@ft.unicamp.br

Recebido: 08/02/2019. **Revisado:** 15/05/2019. **Aceito:** 15/06/2019.

Resumo

O algoritmo *Online Sequential Extreme Learning Machine* é adequado para previsão de Fluxos de Dados com Desvios de Conceito. No entanto, esse tipo de previsão exige implementações de alto desempenho devido à alta taxa de entrada de amostras. Neste trabalho, analisamos implementações paralelas para o *Online Sequential Extreme Learning Machine* em linguagem de programação C, com as bibliotecas OpenBLAS, Intel MKL e MAGMA. A OpenBLAS e a Intel MKL fornecem funções que exploram os recursos *multithread* em CPUs com vários núcleos, o que estende o paralelismo para arquiteturas de multiprocessadores. Por sua vez, a MAGMA oferece funções que são executadas em paralelo em arquiteturas heterogêneas ou híbridas, como sistemas com processadores Multicore e unidades de processamento gráfico, a GPU. Assim, o objetivo deste trabalho é comparar o desempenho – erro de previsão/precisão e tempo real de processamento do fluxo – das implementações em C com o *Online Sequential Extreme Learning Machine* original no MATLAB, ao prever concentrações de material particulado no ar. Os resultados experimentais mostraram que, na maioria dos casos abordados aqui, pelo menos uma das implementações na linguagem C obteve melhor desempenho em relação ao tempo de processamento do fluxo, quando comparado com a versão de referência do MATLAB, executando até 7 vezes mais rápido.

Palavras-Chave: Computação de Alto Desempenho; Desvio de Conceito; Fluxo de Dados

Abstract

The Online Sequential Extreme Learning Machine algorithm is suitable for forecasting Data Streams with Concept Drifts. Nevertheless, forecasting requires high-performance implementations due to the high incoming samples rate. In this work, we analyzed parallel implementations for the Online Sequential Extreme Learning Machine in the C programming language, with OpenBLAS, Intel MKL, and MAGMA libraries. Both OpenBLAS and Intel MKL provide functions that explore the multithread features in multicore CPUs, which expands the parallelism to multiprocessors architectures. In turn, MAGMA offers functions that run in parallel in heterogeneous/hybrid architectures, like Multicore systems with graphics processing unit, the GPU. Thus, the goal of this work is to compare the performance – prediction error/precision and real stream processing time – of the C implementations with the original Online Sequential Extreme Learning Machine in MATLAB when forecasting concentrations of Particulate Matter in the air. Experimental results showed that in most cases approached here, at least one of the C implementations obtained better performance, regarding stream processing time, when compared with the reference MATLAB version, performing up to 7-fold faster.

Key words: Concept Drift; Data Stream; High-Performace Computing

1 Introdução

Os algoritmos de Mineração de Dados e Aprendizado de Máquina possibilitam a extração de várias informações implícitas na grande quantidade de conjuntos de dados disponíveis hoje em dia. Dentre as várias aplicações disponíveis, a utilização desses algoritmos podem ajudar as agências ambientais a gerenciar a qualidade do ar, através do monitoramento e previsão dos poluentes.

A ocorrência de vários episódios agudos de poluição do ar em áreas urbanas têm chamado a atenção de agências ambientais para a importância de se prever a concentração desses poluentes. Isso se justifica porque as altas taxas de poluição atmosférica causam efeitos nocivos à saúde e até mesmo mortes prematuras nos grupos considerados sensíveis, compostos por idosos, crianças e pessoas com doenças cardíacas e respiratórias (CETESB; 2016; Souza et al.; 2015; Vong et al.; 2014).

Os dados de concentrações de poluição do ar obtidos periodicamente podem ser considerdos como Séries Temporais. Além disso, por se tratar de uma sequência ordenada de amostras que chegam a uma alta velocidade, essas séries temporáis também podem ser tratadas como Fluxos de Dados ou, em outras palavras, um número elevado de amostras obtidas por sensores por unidade de tempo.

Considerando essa característica, isto é, produção de dados em grande volume e grande velocidade, os sistemas de previsão em fluxos de dados não conseguem lidar com um número muito grande de amostras na memória principal por um longo período (Cavalcante and Oliveira; 2015), já que isso pode causar uma sobrecarga na memória. Portanto, são necessários sistemas para processamento de dados em tempo real para lidar com previsão de séries temporais em fluxos de dados.

Os fluxos de dados possuem uma natureza dinâmica, ou seja, as distribuições de dados subjacentes desses fluxos tendem a mudar ao longo do tempo. Esse fenômeno, conhecido na literatura como Desvio de Conceito, consiste numa alteração da relação entre as entradas e a variável alvo a ser prevista. Essa alteração pode ocorrer ao longo do tempo (Gama et al.; 2014). As abordagens de aprendizado *online* podem lidar com os desvios de conceito, devido à capacidade de aprender novos padrões de dados ao longo do tempo. Algoritmos como o *Online Sequential Extreme Learning Machine* (OS-ELM) (Liang et al.; 2006), possuem a habilidade de atualizar o modelo de previsão de acordo com a chegada das novas amostras. Essas características tornam o OS-ELM capaz de lidar com desvios de conceito de forma implícita. Isso torna esse algoritmo adequado para previsões em fluxos de dados.

Considerando a relevância do algoritmo OS-ELM, este artigo apresenta uma comparação do desempenho, considerando os erros de previsão/precisão e tempo real de processamento do fluxo, das implementações na linguagem C com a versão original do *Online Sequential Extreme Learning Machine* implementado no software MATLAB, ao prever concentrações de material particulado no ar. O objetivo é verificar em quais situações a implementação de versões paralelas do algoritmo na linguagem C é mais vantajosa. Dessa forma, torna-se mais fácil a incorporação desse algoritmo em outros sistemas independentes de ambientes de simulação, como o MATLAB.

O restante deste artigo está organizado da seguinte forma. A Seção 2 apresenta trabalhos relacionados com o OS-ELM, que abordam sua melhoria de desempenho e também casos aplicados na previsão de poluição atmosférica; A Seção 3 apresenta as estruturas do modelo de aprendizagem ELM e OS-ELM. A Seção 4 discute as nossas implementações em detalhes. Por sua vez, a Seção 5 descreve a metodologia adotada, incluindo a descrição do conjunto de dados, as etapas de pré-processamento e a configuração experimental. Finalmente, a Seção 6 apresenta os resultados experimentais e a Seção 7 discute possíveis direções para trabalhos futuros.

2 Trabalhos Relacionados

Atualmente, vários pesquisadores têm se dedicado a estudar a previsão de Material Particulado menor do que 10 micrômetros (MP_{10}), que são as partículas inaláveis, e outros poluentes do ar. Em Souza et al. (2015), os autores desenvolveram um agrupamento (*ensemble*) de Redes Neurais Artificiais (RNAs) para prever as concentrações diárias de MP_{10}, que apresentaram melhor desempenho quando comparadas às RNAs únicas. Em um trabalho recente Vong et al. (2014), os autores compararam o modelo *Extreme Learning Machine* (ELM) (Huang et al.; 2006) com modelo *Support Vector Machine* (SVM) para a tarefa de classificação num conjunto de dados de concentração de MP_{10}. Os resultados mostraram que o modelo ELM superou o modelo SVM em precisão e tempo de processamento.

No trabalho de Bueno et al. (2017), os autores compararam as abordagens de aprendizagem sequencial e *online* baseadas no algoritmo OS-ELM com abordagens *offline* baseadas no algoritmo ELM original ao prever concentrações de MP_{10}. Os resultados mostraram que o algotimo OS-ELM oferece uma melhor precisão em tais cenários e também que *ensembles* de OS-ELMs melhoram a estabilidade dos resultados. No entanto, de maneira geral, a maioria das abordagens OS-ELM e seus respectivos *ensembles* levaram mais tempo que as abordagens ELM para realizar as previsões.

Cavalcante and Oliveira (2015) implementaram os algoritmos ELM e OS-ELM combinados com os detectores de desvios *Drift Detection Method* (DDM) e *EWMA for Concept Drift Detection* (ECDD). A ideia principal nesse trabalho foi treinar novamente o algoritmo ELM toda vez que ocorre um desvio de conceito e desativar a operação de atualização do algoritmo OS-ELM em períodos de estabilidade.

Os resultados mostraram que as abordagens baseadas no algoritmo OS-ELM obtiveram menor erro de previsão ao prever conjuntos de dados reais. Nesses casos, o algoritmo OS-ELM original apresentava menor erro em dois terços dos conjuntos de dados experimentais, enquanto que, para os conjuntos de dados restante, a combinação OS-ELM e DDM funcionava melhor. Em relação ao tempo de processamento, a combinação dos algoritmos OS-ELM e ECDD acelera 1,5 vezes em alguns casos com conjuntos de dados reais quando comparado

ao algoritmo OS-ELM original e à combinação dos algoritmos OS-ELM e DDM.

No trabalho de Krawczyk (2016) é proposta uma abordagem que combina o algoritmo OS-ELM com o algoritmo *Adaptive Windowing* (ADWIN) e faz a aceleração dos cálculos matriciais em Unidades de Processamento Gráfico (GPU). O autor fez os testes no ambiente R com o RMOA, usando a biblioteca cuBLAS e os seguintes parâmetros: de 10 a 100 nós ocultos, blocos de atualização de 2.500 instâncias e função de ativação sigmoide. Os resultados mostraram uma redução de quase 10 vezes no tempo de processamento.

Baseando-se nos trabalhos levantados, pode-se afirmar que o algoritmo OS-ELM é adequado para prever as concentrações de MP_{10}. No entanto, esse algoritmo gasta mais tempo em comparação com os modelos *offline* do ELM. Apesar disso, também observamos que foi possível reduzir esse impacto com a aceleração em paralelo do OS-ELM, que mostrou resultados promissores.

Neste trabalho, utilizamos implementações paralelas do OS-ELM que executam em processadores de múltiplos núcleos (*multicore*) e em arquiteturas híbridas – que combinam processadores e GPUs – para comparar o desempenho com a versão de referência em MATLAB, utilizada como *benchmark* ao prever concentrações de MP_{10} num fluxo de dados simulado. Dessa forma, pretendemos verificar se essas implementações do algoritmo OS-ELM são adequadas para realizar previsões de MP_{10} em fluxo de dados e também se nossas implementações podem melhorar o desempenho da versão em MATLAB. Escolhemos a implementação em MATLAB como referência, uma vez que esse software é amplamente utilizado pela comunidade científica.

3 Extreme Learning Machines

O algoritmo *Extreme Learning Machine* é uma estrutura de aprendizagem eficiente derivada de uma Rede *Feedforward* de Camada Única (em inglês *Single hidden Layer Feedforward Networks* – SLFN) (Huang et al.; 2006). Usando pesos aleatórios entre a camada de entrada e a camada oculta, o modelo ELM é treinado mais rápido que as RNAs convencionais, que precisam ajustar seus pesos para cada mudança de camada.

Consideramos um conjunto de dados de treinamento $\mathbf{D}_{train} = \{(\mathbf{x}_t, \mathbf{y}_t) | \mathbf{x}_t \in \mathbf{R}^m, \mathbf{y}_t \in \mathbf{R}^n, t = 1, \ldots, T\}$ em que $\mathbf{x}_t$ é uma matriz de dados de entrada, e $\mathbf{y}_t$ é o vetor de variáveis alvo a serem previstas. Um modelo ELM padrão com $L \leq T$ neurônios ocultos e função de ativação $g(x)$, que pode aproximar as T instâncias de $\mathbf{D}_{train}$ com erro zero, é modelado matematicamente de acordo com a Eq. 1 (Huang et al.; 2006):

$$f_L(\mathbf{x}_t) = \sum_{j=1}^{L} \beta_j g(\mathbf{a}_j \cdot \mathbf{x}_t + b_j) = \mathbf{y}_t \quad \text{para } t = 1, \ldots, T. \quad (1)$$

em que $\mathbf{a}_j = [a_{j_1}, a_{j_2}, \ldots, a_{j_r}]^T$ é o vetor com pesos aleatórios que conecta os nós de entrada r com o j-ésimo nó oculto, $j = 1, \ldots, L$; $\beta_j = [\beta_1, \beta_2, \ldots, \beta_j]$ é o vetor de pesos que conecta o j-ésimo nó oculto com os nós de saída; b_j é o *bias* do j-ésimo nó oculto; e $a_j \cdot x_t$ denota o produto interno de a_j por x_t.

Huang et al. (2006) reescreveu as T equações anteriores de uma maneira compacta, de acordo com as equações 2, 3 e 4 a seguir.

$$\mathbf{H}\beta = \mathbf{Y}, \quad (2)$$

$$\mathbf{H} = \begin{bmatrix} g(\mathbf{a}_1 \cdot \mathbf{x}_1 + b_1) & \cdots & (\mathbf{a}_L \cdot \mathbf{x}_1 + b_L) \\ \vdots & \ddots & \vdots \\ g(\mathbf{a}_1 \cdot \mathbf{x}_T + b_1) & \cdots & (\mathbf{a}_L \cdot \mathbf{x}_T + b_L) \end{bmatrix}_{T \times L}, \quad (3)$$

$$\beta = \begin{bmatrix} \beta_1^T \\ \vdots \\ \beta_L^T \end{bmatrix}_{L \times m} \quad e \quad \mathbf{Y} = \begin{bmatrix} \mathbf{y}_1^T \\ \vdots \\ \mathbf{y}_T^T \end{bmatrix}_{T \times n}, \quad (4)$$

em que $\mathbf{Y}$ é a saída que contém as variáveis alvo; $\mathbf{H}$ é a matriz de saída da camada oculta da rede neural, onde a j-ésima coluna de $\mathbf{H}$ é a saída dos j-ésimos nós ocultos em relação às entradas $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_T$; e a t-ésima linha de $\mathbf{H}$ é o vetor de saída da camada oculta em relação a $\mathbf{x}_t$.

Como o algoritmo ELM atribui aleatoriamente os pesos e os viéses (*biases*) de entrada, a geração do modelo de previsão a partir do conjunto de dados de treinamento é baseada na solução de vetor β em função da matriz $\mathbf{H}$ e dos valores y conhecidos, conforme a Eq. 2.

Assim, no conjunto de dados de teste, podemos usar o β calculado para fazer a previsão e comparar o erro com a saída conhecida. A solução para o β pode ser determinada com o Método dos Mínimos Quadrados de acordo com a Eq. 5:

$$\hat{\beta} = \mathbf{H}^{\dagger}\mathbf{Y}, \quad (5)$$

em que $\mathbf{H}^{\dagger}$ é a inversa generalizada de $\mathbf{H}$.

Existem várias maneiras de calcular o inverso generalizado de Moore-Penrose de uma matriz, como o Método de Projeção Ortogonal e a Decomposição de Valor Singular (SVD), entre outros. Neste trabalho, vamos considerar o método de projeção ortogonal que o algoritmo OS-ELM usa, apresentado pela Eq. 6:

$$\mathbf{H}^{\dagger} = (\mathbf{H}^T\mathbf{H})^{-1}\mathbf{H}^T \quad (6)$$

em que $\mathbf{H}^T$ é a transposta $\mathbf{H}$ e $\mathbf{H}^T\mathbf{H}$ é não-singular.

Para obter uma solução mais estável, evitando uma $\mathbf{H}^T\mathbf{H}$ singular ou mal dimensionada, nós adicionamos um parâmetro de corte ($\frac{1}{\lambda}$) a sua diagonal, como feito em Huang et al. (2012) e em Krawczyk (2016). Assim, nós podemos calcular o β de acordo com a Eq. 7:

$$\beta = (\frac{\mathbf{I}}{\lambda} + \mathbf{H}^T\mathbf{H})^{-1}\mathbf{H}^T\mathbf{Y} \quad (7)$$

em que $\mathbf{I}$ é uma matriz identidade do tamanho de $\mathbf{H}$. O modelo ELM é resumido no Algoritmo 1 a seguir.

Algoritmo 1: Modelo de aprendizagem ELM

Entrada: Conjunto de dados de treinamento $\mathbf{D}_{train} = (\mathbf{x}_t, \mathbf{y}_t)_{t=1}^{T}$; número de nós ocultos (L, em que $L \leq T$); função de ativação $g(x)$;
Saída: β, erro de previsão; tempo de treinamento;

1 **início**
2 Gere pesos de entrada $\mathbf{a}_j$ e *biases* b_j aleatórios;
3 Calcule a matriz da camada oculta $\mathbf{H}$ with $\mathbf{D}_{train}$ de acordo com a Eq. 3;
4 Calcule o vector β com os pesos de saída, de acordo com a Eq. 7;
5 Calcule o tempo de treinamento e o erro de previsão;
6 **fim**

3.1 Online Sequential Extreme Learning Machine

O trabalho de Liang et al. (2006) propôs o algoritmo *Online Sequential Extreme Learning Machine* (OS-ELM), uma variante do algoritmo ELM padrão. O OS-ELM possui os mesmos parâmetros de entrada do ELM, mais o número do conjunto de treinamento inicial N_0 e o tamanho do bloco para a fase de aprendizagem sequencial N_k.

Na fase de treinamento inicial, o OS-ELM usa um conjunto de treinamento $\mathbf{D}_0 = (\mathbf{x}_t, \mathbf{y}_t)_{t=1}^{N_0} \subset \mathbf{D}_{train}$ (com $N_0 < T$) para construir o modelo inicial. Na fase de aprendizado *online* e sequencial, o algoritmo OS-ELM usa apenas as novas instâncias, que chegam bloco a bloco, para atualizar o modelo.

A etapa de inicialização do OS-ELM é igual a do ELM. A Eq. 8 determina o vetor de pesos de saída β_0:

$$\beta_0 = (\mathbf{H}_0^T\mathbf{H}_0)^{-1}\mathbf{H}_0^T\mathbf{Y}_0 \tag{8}$$

em que $\mathbf{Y}_0 = [y_1, y_2, \ldots, y_T]^{N_0}$ é a saída de $\mathbf{D}_0$ e $\mathbf{H}_0$ é a matriz oculta inicial obtida com $\mathbf{D}_0$. Para fazer $\mathbf{H}_0^T\mathbf{H}_0$ não-singular ($rank(\mathbf{H}_0) = L$), é necessário $N_0 \geq L$ (Liang et al.; 2006).

Podemos reescrever a Eq. 8 como $\beta_0 = \mathbf{M}_0\mathbf{H}_0^T\mathbf{Y}_0$, em que $\mathbf{M}_0$ é a matriz de covariância inicial, calculada pela Eq. 9:

$$\mathbf{M}_0 = (\mathbf{H}_0^T\mathbf{H}_0)^{-1} \tag{9}$$

No estágio de aprendizado *online* e sequencial, quando um novo bloco chega, o algoritmo OS-ELM calcula os novos pesos de saída β_{k+1} usando conceitos do algoritmo *Recursive Least Squared* (RLS), de acordo com as equações 10, 11, 12 e 13. O trabalho de Liang et al. (2006) fornece uma explicação detalhada das equações 12 e 13.

$$\mathbf{H}_{k+1} = \begin{bmatrix} g(\mathbf{a}_1 \cdot \mathbf{x}_{(\sum_{l=0}^{k} T_l)+1} + b_1) & \cdots & (\mathbf{a}_L \cdot \mathbf{x}_{(\sum_{l=0}^{k} T_l)+1} + b_L) \\ \vdots & \ddots & \vdots \\ g(\mathbf{a}_1 \cdot \mathbf{x}_{\sum_{l=0}^{k+1} T_l} + b_1) & \cdots & (\mathbf{a}_L \cdot \mathbf{x}_{\sum_{l=0}^{k+1} T_l} + b_L) \end{bmatrix}_{T_{k+1} \times L}, \tag{10}$$

$$\mathbf{Y}_{k+1} = \begin{bmatrix} \mathbf{y}^T_{(\sum_{l=0}^{k} T_l)+1} \\ \vdots \\ \mathbf{y}^T_{\sum_{l=0}^{k+1} T_l} \end{bmatrix}_{T_{k+1} \times n}, \tag{11}$$

$$\mathbf{M}_{k+1} = \mathbf{M}_k - \mathbf{H}_{k+1}^T(1 + \mathbf{H}_{k+1}\mathbf{M}_k\mathbf{H}_{k+1}^T)^{-1}\mathbf{H}_{k+1}\mathbf{M}_k, \tag{12}$$

$$\beta_{k+1} = \beta_k + \mathbf{M}_{k+1}\mathbf{H}_{k+1}^T(\mathbf{Y}_{k+1} - \mathbf{H}_{k+1}\beta_k). \tag{13}$$

Podemos observar que o treinamento dos modelos ELM e OS-ELM se utilizam de várias operações matriciais. Essas operações podem se tornar muito custosas em termos computacionais ao lidar com um conjunto de dados muito grande, um caso típico de fluxo de dados. Vale lembrar que a própria dinâmica dos fluxos de dados impõe uma restrição de tempo para a execução do algoritmo. Portanto, o maior foco neste artigo é no ganho de tempo desses algoritmos.

4 Implementações Paralelas

Para adaptar o algoritmo OS-ELM em MATLAB (Huang; 2013) e nossas implementações em linguagem C para processamento de fluxo de dados, consideramos um conjunto de dados em que um pedaço de tamanho N_0 é utilizado para o treinamento inicial, e as instâncias restantes simulam um fluxo *online*, que será previsto pelo modelo OS-ELM. Também consideramos o tempo de atualização e previsão da fase *online* do modelo OS-ELM para avaliar o desempenho, conforme o Algoritmo 2.

Além da implementação em MATLAB, referenciada como OS-ELM, desenvolvemos três implementações distintas em linguagem C: OS-ELM com OpenBLAS (OS-ELMob), OS-ELM com MKL (OS-ELMmkl) e OS-ELM com MAGMA (OS-ELMmgm). OpenBLAS, Intel MKL e MAGMA são bibliotecas que nos permitem executar as operações matriciais com as funções do *Basic Linear Algebra Subprograms* (BLAS) e do *Linear Algebra PACKage* (LAPACK).

As funções do OpenBLAS (Zhang et al.; 2012) e do Intel MKL executam em ambientes *multithread*, o que permite o ganho de tempo com paralelismo em processadores *multicore* e também em arquiteturas de multiprocessadores. O MATLAB aplica paralelismo *multithread* nas funções de álgebra linear e numéricas por padrão, desde a versão 2008a (MathWorks; 2017). Por sua vez, as funções do MAGMA (Tomov et al.; 2010) exploram o paralelismo em arquiteturas heterogêneas/híbridas para sistemas que combinam processadores *multicore* e GPU. Neste trabalho, as implementações em linguagem C ficam restritas aos casos de regressão com função de ativação sigmoide ($g(x) = \frac{1}{(1+e^{(-x)})}$).

Em todas as implementações em C, usamos as respectivas funções `dgetrf()` e `dgetri()` para calcular a matriz de covariância inicial $\mathbf{M}_0$ (passo 5 do Algoritmo 2). As respectivas funções `dgesv()` foram usadas para encontrar β_0 (passo 6 do Algoritmo 2), onde também foi adicionando um parâmetro de corte $\frac{1}{\lambda}$, conforme a Eq. 7, em que $\lambda = 50\epsilon$, e ϵ é a precisão da máquina para o tipo real (*float*).

Algoritmo 2: OS-ELM para Fluxo de Dados

Entrada: Fluxo de dados $\mathbf{D} = (\mathbf{x}_t, \mathbf{y}_t)_{t=1}^{T}$; nº de nós ocultos ($L$, em que $L \leq T_0 < T$); função de ativação $g(x)$; nº de instâncias para o treinamento inicial N_0; tamanho do bloco para a fase de aprendizado *online* e sequencial N_k;

Saída: Precisão do treinamento inicial e da previsão do fluxo; Tempo de processamento do treinamento inicial e do processamento do fluxo; Tempo médio de atualização e previsão dos blocos;

1 **início**
2 Considere $\mathbf{D}_0(\mathbf{x}_t, \mathbf{y}_t)_{t=1}^{N_0}$ para o treino inicial;
3 Gere pesos de entrada $\mathbf{a}_j$ e *biases* b_j aleatórios;
4 Calcule $\mathbf{H}_0$, com $\mathbf{D}_0$ ($\approx$ Eq. 3);
5 Calcule $\mathbf{M}_0$, de acordo com a Eq. 9;
6 Calcule o vetor β_0 através da Eq. 8;
7 Calcule o tempo e a precisão do treino inicial;
8 **para** $k = N_0; k \leq T; k = k + N_k$ **faça**
9 Calcule $\mathbf{H}_{k+1}$ conforme a Eq. 10;
10 Calcule o vetor de previsões $\mathbf{P}_{k+1}$ de $\mathbf{D}_{k+1}$ com o β_k previamente calculado;
11 Defina $\mathbf{Y}_{k+1}$ de acordo com a Eq. 11;
12 Atualize a $\mathbf{M}_{k+1}$ de acordo com a Eq. 12;
13 Atualize o β_{k+1} de acordo com a Eq. 13;
14 **fim**
15 Calcule o tempo real de processamento e o erro de previsão no fluxo;
16 **fim**

Na fase sequencial e *online*, nós usamos as correspondentes funções `dgesv()` para encontrar a matriz inversa no cálculo de $\mathbf{M}_{k+1}$ (passo 12 do Algoritmo 2) ao invés das funções `dgetrf()` e `dgetri()`, a fim de melhorar o desempenho, como recomendado pelas documentações das bibliotecas.

As implementações OS-ELMob e OS-ELMmkl se diferenciam apenas na função que copia e gera a transposta das matrizes, chamada `cblas_domatcopy()` no OpenBLAS e `mkl_domatcopy()` no MKL. No OS-ELMmgm, foram feitas mais modificações, já que o MAGMA possui suas próprias funções com diferentes parâmetros.

Na implementação OS-ELMmgm nós adaptamos os passos 4 a 13 do Algoritmo 2 usando as funções do MAGMA. Nesta implementação, também foi necessário transferir os dados para a memória da GPU depois que eles foram carregados. A geração de pesos e *biases* aleatórios (passo 3 do Algoritmo 2) usaram a função `LAPACKE_dlarnv` em todas as implementações em linguagem C, já que o MAGMA não possui uma função `dlarnv()` correspondente.

5 Materiais e Métodos

Nesta seção, descrevemos os métodos e dados empregados para a avaliação de desempenho. Primeiro, explicamos o conjunto de dados utilizado e as etapas de pré-processamento. Depois, descrevemos a configuração das implementações OS-ELM. O objetivo dos testes é avaliar se o algoritmo OS-ELM é adequado para previsão em fluxos de dados com concentrações de MP_{10} e se as nossas implementações melhoram o seu desempenho.

5.1 O conjunto de dados e as etapas de pré-processamento

Realizamos os experimentos com um conjunto de dados ambientais que contém amostras de concentrações de MP_{10}, medidas em micrômetros por metro cúbico ($\mu g/m^3$), coletadas sequencialmente ao longo do tempo. Esse conjunto de dados, fornecido pelo sistema QUALAR (CETESB; 2019), contém amostras horárias coletadas de 01 de janeiro de 1998 a 23 de novembro de 2017, da estação automática de Cubatão – Vila Parisi. São um total de 174.397 amostras, zero como valor mínimo e 1.470 $\mu g/m^3$ como valor máximo.

Entre as estações monitoradas pela Companhia Ambiental do Estado de São Paulo (CETESB), a estação de Cubatão – Vila Parisi apresentou o maior número de amostras que excedem os padrões de qualidade do ar do Estado de São Paulo, no que se refere a MP_{10}. Cerca de 1.657 medições ultrapassaram os limites dentro do período estudado (CETESB; 2019).

Para mitigar os efeitos indesejáveis das 8.011 amostras faltantes que esse conjunto de dados apresenta, nós usamos o software Amelia II para preenche-las com o valor mais provável (Honaker et al.; 2011). Para os casos não resolvidos pelo Amelia II, optamos por preencher com uma interpolação linear.

Também fizemos uma análise de *outliers* em que normalizamos 964 amostras acima de 350 $\mu g/m^3$ (valor máximo da concentração de MP_{10} reportado pela CETESB para a estação de Cubatão – Vila Parisi). Também usamos o filtro Hampel para a detectar e substituir os demais *outliers* presentes no conjunto. Após essas etapas, normalizamos todos os dados do conjunto entre $[0, 1]$ pela normalização min-max, como sugerido por Huang (2013) e aplicado por Vong et al. (2014).

Na última etapa de pré-processamento, organizamos uma série temporal composta pela amostra atual mais as amostras dos últimos cinco instantes de tempo $x_n = [s_{n-5} \ldots, s_{n-1}, s_n]$, que foram usadas para prever a amostra do próximo instante $y = [s_{n+1}]$. Ou seja, utilizamos amostras das ultimas seis horas para prever o valor da próxima hora.

5.2 Configurações dos algoritmos OS-ELM implementados

Testamos casos com 25, 50 e 100 neurônios ocultos (NO), valores dentro do intervalo utilizado por Krawczyk (2016), e blocos de atualização de 1, 10, 100, 1.000, 2.500 e 5.000 instâncias. Para o conjunto de treinamento inicial, utilizamos as 5.000 instâncias mais antigas e simulamos um fluxo de dados com as 169.397 instâncias mais recentes.

Para cada caso testado, avaliamos o desempenho medindo os erros de previsão e os tempos de processamento do fluxo simulado através das médias

e desvios-padrão. Como medida de erro de previsão, consideramos a Raiz do Erro Quadrático Médio (RMSE) entre as saídas reais e previstas. Em relação ao tempo, consideramos os tempos reais para processar todo o fluxo simulado. Os experimentos foram repetidos em 50 tentativas e conduzidos em uma máquina virtual NC6 hospedada na nuvem Microsoft Azure com a seguinte configuração:

- 6 vCPUs Xeon E5-2690 @ 2,60GHz;
- 56 GB de Memória RAM;
- 1 Tesla K80, com *clock* de 823,5 MHz e 11,5 GB;
- Sistema Operacional CentOS 7.3 HPC 64 bits;
- Software MATLAB versão 9.3 (R2017b) *trial*;
- OpenBLAS 0.2.20 e Intel MKL 2018.0.1;
- MAGMA 2.2.0 configurado para o CUDA 9.0 e para o OpenBLAS 0.2.20.

Também calculamos os p-valores para as 50 repetições em cada caso usando o teste de Shapiro-Wilk. Os casos cujos p-valores foram maiores que 0,05 estão sinalizados em itálico nas tabelas que apresentam os resultados.

6 Resultados Experimentais

A Tab. 1 mostra os RMSEs médios e o Desvio Padrão (DP) do fluxo simulado ao variar o número de NOs e o Tamanho dos Blocos (TB). Os menores RMSEs estão destacados em fonte mais escura.

O cenário apresentado na Tab. 1 indica que, apesar da pequena diferença (10^{-5}), os casos com menores RMSEs ocorrem com 100 neurônios ocultos e com TBs de 1 e 10 instâncias na implementação OS-ELMob. Além disso, na maioria dos casos testados, o RMSE das implementações em linguagem C (OS-ELMob, OS-ELMmkl e OS-ELMmgm) foram menores que a apresentada pelo OS-ELM (de 10^{-2} a 10^{-5}), sendo que as diferenças aumentavam conforme os TBs também aumentavam. Portanto, concluímos que o uso das funções `dgesv()` e a adoção do parâmetro de corte no treinamento inicial das implementações em linguagem C mostraram resultados mais estáveis. Na Tab. 1 todos os casos foram estatisticamente significativos (p-valor<0,05).

A Tab. 2 apresenta as médias e os DPs dos tempos reais de processamento do fluxo simulado, comparando OS-ELM, o OS-ELMob e o OS-ELMmkl, que são implementações *multithread* e o OS-ELMmgm, executado em arquitetura híbrida. Não fizemos configurações adicionais relacionadas ao número de *threads*. Portanto, todas as implementações foram executadas com seus respectivos valores padrão: 6 *threads*, um para cada núcleo virtual disponível.

Quanto aos tempos observados na Tab. 2, podemos notar que a maioria dos casos foram estatisticamente significativos, exceto para a implementação OS-ELMmkl. A Tab. 2 mostra que as implementações OS-ELMob e OS-ELMmkl executaram mais rápido que a implementação OS-ELM em TBs de até 100 instâncias. No entanto, para TBs com 5.000 instâncias, a implementação OS-ELMob demorou 2,3 vezes e o OS-ELMmkl demorou 1,7 vezes a mais que o OS-ELM, em casos com 100 NOs.

Em contraste com o OS-ELMob e o OS-ELMmkl, a Tab. 2 demonstra que o OS-ELMmgm foi mais rápido para TBs de 2.500 e 5.000 instâncias. Para TBs de 1, 10 e 100, o OS-ELMmgm foi o mais lento. Em TBs de 1.000, o OS-ELMmgm superou o OS-ELMob e o OS-ELMmkl, e quase alcançou o mesmo tempo que o OS-ELM no caso com 100 NOs.

A Tab. 2 também mostra que, nos casos com TBs de 1.000 instâncias nenhuma das implementações em C foi mais rápida do que o OS-ELM. Apesar disso, mesmo a implementação mais lenta poderia processar um fluxo de dados em tempo real com uma taxa de chegada de até 1.000 instâncias por segundo em qualquer um dos casos testados.

Para visualizar melhor as acelerações das implementações em C comparadas com as implementações OS-ELM, as figuras 1, 2 e 3 apresentam as acelerações no processamento do fluxo, com 25, 50 e 100 nós ocultos, respectivamente. Além disso, as figuras 1, 2 e 3 mostram que as implementações OS-ELMob e OS-ELMmkl apresentaram melhores desempenhos para TBs de até 100 instâncias, que possuem mais iterações no laço **para** (passos 8 ao 14) do Algoritmo 2, e operações matriciais menos complexas.

Tabela 1: RMSEs do fluxo simulado para cada caso.

NO	TB	OS-ELM	OS-ELMob	OS-ELMmkl	OS-ELMmgm
25	1	0,10831	0,10830	0,10830	0,10830
	10	0,10832	0,10833	0,10831	0,10831
	100	0,10839	0,10831	0,10832	0,10832
	1.000	0,10855	0,10836	0,10836	0,10836
	2.500	0,10947	0,10840	0,10839	0,10839
	5.000	0,11109	0,10841	0,10841	0,10839
50	1	0,10819	0,10820	0,10819	0,10819
	10	0,10821	0,10820	0,10819	0,10820
	100	0,10828	0.10822	0,10822	0.10821
	1.000	0,10843	0,10826	0,10826	0,10826
	2.500	0,10938	0,10829	0,10830	0,10831
	5.000	0,11100	0,10830	0,10831	0,10832
100	**1**	0,10802	**0,10799**	0,10800	0,10800
	10	0,10801	**0,10799**	0,10800	0,10801
	100	0,10809	0,10801	0,10802	0,10803
	1.000	0,10824	0,10806	0,10807	0,10807
	2.500	0,10921	0,10811	0,10810	0,10811
	5.000	0,11086	0,10813	0,10814	0,10814

[1] O DP de todos os casos variou entre 2×10^{-5} e 6×10^{-5}.

Tabela 2: Tempos reais (em segundos) de processamento do fluxo

NO	TB	OS-ELM	OS-ELMob	OS-ELMmkl	OS-ELMmgm
25	1	*4,19 ± 0,06*	1,11 ± 0,01	**0,55 ± 0,02**	*118,57 ± 1,10*
	10	1,44 ± 0,04	0,41 ± 0,07	**0,25 ± 0,01**	13,74 ± 0,34
	100	0,92 ± 0,08	**0,74 ± 0,03**	0,74 ± 0,07	2,59 ± 0,12
	1.000	**3,65 ± 0,23**	8,85 ± 0,18	5,48 ± 0,21	4,73 ± 0,19
	2.500	*20,71 ± 0,80*	38,15 ± 0,22	*26,5 ± 0,14*	**16,74 ± 0,54**
	5.000	*51,69 ± 0,22*	*124,20 ± 0,21*	*91,53 ± 0,26*	**37,40 ± 0,62**
50	1	7,13 ± 0,46	2,50 ± 0,02	**0,97 ± 0,01**	*124,06 ± 1,00*
	10	*2,55 ± 0,23*	0,73 ± 0,03	**0,45 ± 0,01**	14,40 ± 0,29
	100	1,1 ± 0,12	0,97 ± 0,08	**0,87 ± 0,06**	2,64 ± 0,16
	1.000	**3,80 ± 0,14**	9,04 ± 0,14	*5,48 ± 0,21*	4,84 ± 0,14
	2.500	*20,80 ± 0,69*	*38,39 ± 0,22*	*26,51 ± 0,21*	**16,8 ± 0,39**
	5.000	52,58 ± 0,28	124,56 ± 0,32	*92,35 ± 0,32*	**37,65 ± 0,59**
100	**1**	18,90 ± 1,05	*8,11 ± 0,05*	***4,48 ± 0,06***	*138,08 ± 1,13*
	10	3,92 ± 0,44	1,79 ± 0,05	***1,59 ± 0,11***	15,81 ± 0,29
	100	1,39 ± 0,16	**1,27 ± 0,06**	1,40 ± 0,08	2,81 ± 0,10
	1.000	***4,12 ± 0,16***	9,45 ± 0,15	*6,46 ± 0,23*	4,90 ± 0,16
	2.500	20,94 ± 0,59	*38,96 ± 0,21*	*27,46 ± 0,18*	**17,28 ± 0,42**
	5.000	53,47 ± 0,23	125,95 ± 3,36	*93,13 ± 0,37*	**37,99 ± 0,53**

[1] Os valores de NO e TB em negrito apresentam as melhores precisões.
[2] Tempos em negrito indicam a versão mais rápida em cada caso.
[3] Os valores destacados em itálico e possuem p-valor>0,05.

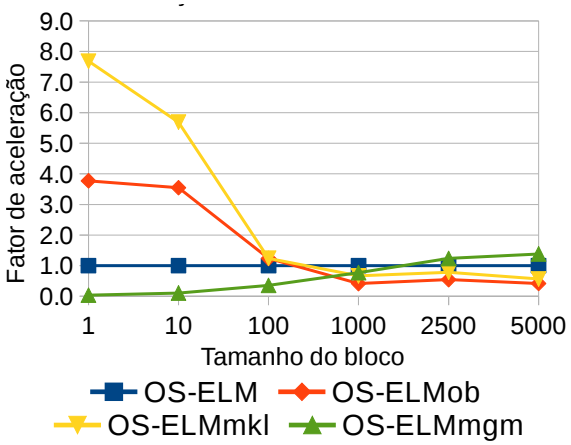

Figura 1: Acelerações com 25 nós ocultos.

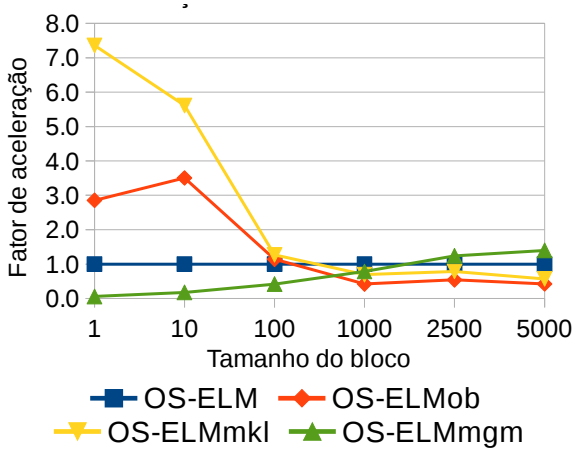

Figura 2: Acelerações com 50 nós ocultos.

Com o OS-ELMmgm ocorre o oposto. Essa implementação vai passando a funcionar melhor com TBs maiores, que possuem menos iterações no laço e operações matriciais mais complexas. No entanto, mesmo para o maior TB analisado, com 5.000 instâncias, a aceleração do OS-ELMmgm foi de apenas cerca de 1,4 vezes comparado com o OS-ELM.

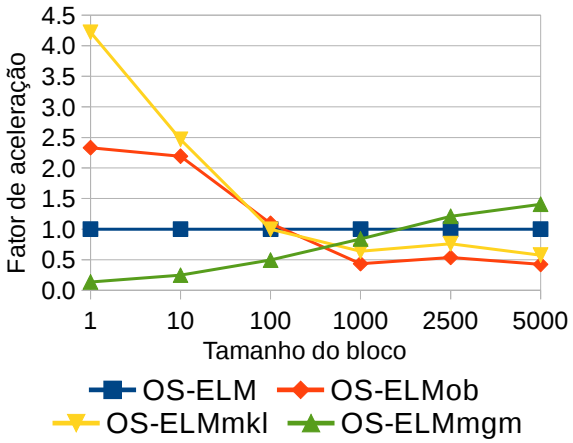

Figura 3: Acelerações com 100 nós ocultos

7 Conclusões

Os resultados mostraram que nossas implementações em C superaram o *benchmark* OS-ELM na maioria dos casos, quando comparamos os tempos reais de processamento do fluxo. Em geral, a implementação OS-ELMmkl apresentou os melhores desempenhos ao usar TBs de 1 e 10 instâncias, acelerando até 7,5 vezes quando comparado ao *benchmark* OS-ELM. A implementação OS-ELMob foi ligeiramente melhor nos casos com TBs de 100, com 25 e 100 NOs.

Nos casos com as melhores precisões – 100 NOs com TBs de 1 e 10 instâncias – a implementação OS-ELMmkl acelerou até 4,2 vezes comparado ao OS-ELM. Na maioria dos casos, com exceção aos com TBs de 1.000, pelo menos uma das abordagens na linguagem C apresentou melhor desempenho quanto aos tempos analisados. O mesmo é valido para

os RMSEs, em que, na maioria dos casos, foram equivalentes ou menores.

Por outro lado, a implementação OS-ELMmgm, usando arquitetura híbrida, apresentou os melhores desempenhos no tempo de processamento ao usar TBs de 2.500 e 5.000 instâncias. No entanto, esse ganho foi de apenas 1,5 vezes. Nós entendemos que esse comportamento está relacionado à sobrecarga de comunicação entre as memórias de CPU e GPU nas funções híbridas do MAGMA. Para blocos menores (TBs até 100 instâncias), devido à grande quantidade de iterações do laço **para** (passos 8 a 14 do Algoritmo 2), o número de chamadas de funções é maior e esse tempo de sobrecarga se torna considerável. Destacamos que isso não significa que obteremos os mesmos resultados em outros sistemas que não estejam na nuvem.

Em cenários com altas taxas de dados de entrada, por exemplo 1.000 amostras por segundo, o uso de *ensembles* OS-ELM pode aproveitar o ganho de desempenho das abordagens apresentadas neste trabalho. Devido aos tempos observados, algumas das implementações aqui apresentadas (OS-ELMob e OS-ELMmkl) também podem ser executadas em processadores embarcados ou de baixo custo.

Com base nos resultados obtidos, acreditamos também que os conceitos e técnicas de alto desempenho apresentados neste trabalho podem ser adequados para a execução dos modelos de aprendizagem OS-ELM em *ensembles*, bem como para outros tipos de modelos de aprendizagem *online*.

Agradecimentos

Os autores gostariam de agradecer ao programa *Microsoft Azure for Research* por fornecer os recursos de computação em nuvem e o suporte da NVIDIA Corporation com a doação da GPU utilizada para testes iniciais. O primeiro autor também agradece ao Instituto Federal de Educação, Ciência e Tecnologia de São Paulo por apoiar sua pesquisa.

Referências

Bueno, A., Coelho, G. P. and Bertini, J. R. (2017). Online Sequential Learning based on Extreme Learning Machines for Particulate Matter Forecasting, *Brazilian Conference on Intelligent Systems (BRACIS)*, IEEE, Uberlandia, pp. 169–174. http://dx.doi.org/10.1109/BRACIS.2017.25.

Cavalcante, R. C. and Oliveira, A. L. I. (2015). An approach to handle concept drift in financial time series based on Extreme Learning Machines and explicit Drift Detection, *Proceedings of the International Joint Conference on Neural Networks*, Vol. 2015-Septe, IEEE, pp. 1–8. http://dx.doi.org/10.1109/IJCNN.2015.7280721.

CETESB (2016). Qualidade do ar no estado de Sao Paulo 2015, *Technical report*, CETESB, São Paulo. Disponível em https://cetesb.sp.gov.br/ar/wp-content/uploads/sites/28/2013/12/RQAR-2015.pdf (Accessado em 8 de Fevereiro de 2019).

CETESB (2019). Qualar | Qualidade do Ar - Sistema Ambiental Paulista - Governo de SP. Disponível em https://cetesb.sp.gov.br/ar/qualar/ (Acessado em 8 de Fevereiro de 2019).

Gama, J., Žliobaitė, I., Bifet, A., Pechenizkiy, M. and Bouchachia, A. (2014). A survey on concept drift adaptation, *Computing Surveys* **46**(4): 1–37. http://dx.doi.org/10.1145/2523813.

Honaker, J., King, G. and Blackwell, M. (2011). Amelia II: A program for missing data, *Journal of Statistical Software* **45**(7): 1–47. http://dx.doi.org/10.18637/jss.v045.i07.

Huang, G.-B. (2013). MATLAB Codes of ELM Algorithm. Disponível em http://www.ntu.edu.sg/home/egbhuang/elm_random_hidden_nodes.html (Acessado em 8 de Fevereiro de 2019).

Huang, G.-B., Zhou, H., Ding, X. and Zhang, R. (2012). Extreme learning machine for regression and multiclass classification, *IEEE transactions on systems, man, and cybernetics. Part B, Cybernetics* **42**(2): 513–29. http://dx.doi.org/10.1109/TSMCB.2011.2168604.

Huang, G.-B., Zhu, Q.-y., Siew, C.-k., Ã, G.-b. H., Zhu, Q.-y., Siew, C.-k., Huang, G.-B., Zhu, Q.-y. and Siew, C.-k. (2006). Extreme learning machine: Theory and applications, *Neurocomputing* **70**(1–3): 489–501. http://dx.doi.org/10.1016/j.neucom.2005.12.126.

Krawczyk, B. (2016). GPU-accelerated extreme learning machines for imbalanced data streams with Concept Drift, *Procedia Computer Science* **80**: 1692–1701. http://dx.doi.org/10.1016/j.procs.2016.05.509.

Liang, N.-Y., Huang, G.-B., Saratchandran, P. and Sundararajan, N. (2006). A Fast and Accurate Online Sequential Learning Algorithm for Feedforward Networks, *IEEE Transactions on Neural Networks* **17**(6): 1411–1423. http://dx.doi.org/10.1109/TNN.2006.880583.

MathWorks (2017). MATLAB Multicore. Disponível em https://www.mathworks.com/discovery/matlab-multicore.html (Acessado em 8 de Fevereiro de 2019).

Souza, R., Coelho, G. P., Silva, A. E. A. and Pozza, S. A. (2015). Using Ensembles of Artificial Neural Networks to Improve PM10 Forecasts, *Chemical Engineering Transactions* **43**: 2161–2166. http://dx.doi.org/10.3303/CET1543361.

Tomov, S., Dongarra, J. and Baboulin, M. (2010). Towards dense linear algebra for hybrid GPU accelerated manycore systems, *Parallel Computing* **36**(5–6): 232–240. http://dx.doi.org/10.1016/j.parco.2009.12.005.

Vong, C. M., Ip, W. F., Wong, P. K. and Chiu, C. C. (2014). Predicting minority class for suspended particulate matters level by extreme learning machine, *Neurocomputing* **128**: 136–144. http://dx.doi.org/10.1016/j.neucom.2012.11.056.

Zhang, X., Wang, Q. and Zhang, Y. (2012). Model-driven level 3 BLAS performance optimization on Loongson 3A processor, *Proceedings of the*

International Conference on Parallel and Distributed Systems – ICPADS pp. 684–691. http://dx.doi.org/10.1109/ICPADS.2012.97.