Gradiente Descendente
Avaliação de estratégias de paralelização

Gustavo Epifânio, Jefferson Colares

[1]CEFET - Centro Federal de Educação Tecnológica Celso Suckow da Fonseca
Rio de Janeiro - RJ - Brasil

gustavo.epifanio@eic.cefet-rj.br, jefferson.colares@eic.cefet-rj.br

Resumo. *O gradiente descendente, devido à sua simplicidade e bom desempenho, é muito utilizado como método de otimização ou minimização em diversas aplicações. A crescente complexidade dessas dessas aplicações, no entanto, exige cada vez maior poder computacional e algoritmos mais eficientes. Para atender essa necessidade, uma das alternativas disponíveis é a paralelização de processos. Nesse trabalho, implementamos algumas das principais abordagens de processamento paralelo do gradiente descendente e comparamos o desempenho delas na tarefa de otimização no cálculo de regressão linear. Nossa pesquisa mostra que a paralelização do algoritmo promove significativa melhoria em seu desempenho e sugere que a paralelização também pode fornecer excelentes resultados em problemas mais complexos.*

1. Introdução

O gradiente descendente é um dos mais populares algoritmos de otimização. Devido à sua simplicidade e bom desempenho, ele é muito utilizado em aplicações de aprendizado de máquina, em especial, no treinamento de redes neurais. A crescente complexidade dessas redes, no entanto, exige cada vez maior poder computacional e algoritmos mais eficientes. Uma das opções disponíveis para melhorar a eficieência computacional é a paralelização de processos, entretanto, o gradiente descendente é considerado um algoritmo difícil de paralelizar, devido à sua natureza sequencial.

Nesse trabalho, implementamos algumas das principais soluções propostas para o processamento paralelo do gradiente descendente e analisamos seu desempenho na tarefa de treinar um modelo de regressão linear utilizando apenas computadores pessoais simples, dotados de procesadores multicore comuns. Nossa pesquisa mostra que a paralelização do gradiente descendente melhora o desempenho e a acurácia do modelo.

1.1. O Gradiente Descendente

O gradiente descendente é um algoritmo de otimização. Sua tarefa é encontrar os valores dos parâmetros que, aplicados a uma determinada função, a faça apresentar o resultado esperado.

Tomando como exemplo a regressão linear, que utiliza a equação reduzida da reta $ax + b = y$, o gradiente seria utilizado para obter os valores de a e b que, aplicados à equação, possibilitem traçar um reta que apresente a menor distância média em relação a todos os pontos de um conjunto de pares x, y fornecidos.

Encontrar os valores ideais de a e b é um processo de tentativa-e-erro. É preciso avaliar a hipótese ($ax + b$) repetidas vezes, utilizando diferentes as e bs, sobre o conjunto

de exemplos x e ir verificando o quão distante o resultado da equação se encontra da resposta correta (y).

Para conhecer a distância entre a hipótese proposta e o valor esperado utiliza-se uma função de custo. A função de custo mais adequada para um modelo de regressão linear é a função de erro quadrático médio: Calcula-se o quadrado da diferença entre o valor estimado para cada exemplo x no conjunto de exemplos X e cada valor esperado y, somam-se todos os resultados obtidos e divide-se pela quantidade de exemplos (m) no conjunto, conforme abaixo:

$$EQM = 1/m * \sum_{i=1}^{m}((ax + b) - y)^2$$

Os valores ideais dos parâmetros são aqueles que apresentam o menor custo. Para encontrá-los, é necessário utilizar a cada tentativa novos valores de parâmetros a e b que nos deixem mais próximos do menor custo. Para encontrar os próximos valores, calcula-se a derivada de cada parâmetro (derivada parcial) em relação ao custo obtido. Esse conjunto de derivadas chama-se gradiente. A cada iteração, atualizam-se os parâmetros calculando o gradiente e subtraindo esse valor do valor obtido na última iteração. Eventualmente, acaba-se chegando ao valor mínimo (convergência). Para aprimorar esse processo, antes da atualização multiplica-se o gradiente por uma constante chamada taxa de aprendizado (α). A representação matemática do algoritimo do Gradiente descendente é:

$$\theta_{(j)} = \theta_{(j)} - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_{(j)}^{(i)}$$

Nesa equação, que deve ser calculada repetidas vezes até encontrar o menor θ, as variáveis são:

- θ: conjunto de parâmetros. No exemplo da equação da reta, seria um conjunto contendo o a e o b.
- j: índice do parâmetro dentro do conjunto de parâmetros θ. Na equação linear, que tem dois parâmetos, j pode ser 0 ou 1.
- α: taxa de aprendizado. Parâmetro que modifica a velocidade de convergência.
- m: quantidade de exemplos(x) no conjunto de exemplos.
- $h_\theta(x)$: hipótese. É o resultado obtido aplicando-se os parâmetros aos valores de X na equação proposta como solução. No caso da regressão linear, a equação da reta.
- i: é o índice do exemplo dentro do conjunto de exemplos X. Vai de 0 a m.
- y: é o valor esperado de retorno para cada exemplo conhecido.

1.2. Tipos de gradiente descendente

Como visto anteriormente, para calcular o gradiente dos parâmetros em relação ao custo, que é calculado em relação a todos os exemplos no conjunto X. Isso nos leva a um problema: quando o conjunto X possui muitas variáveis e muitos exemplos de cada variável, o gradiente descendente pode levar muito tempo até encontrar os parâmetros ideais. Para contornar esse problema, existem outras versões do gradiente descendente além da versão simples, descrita acima.

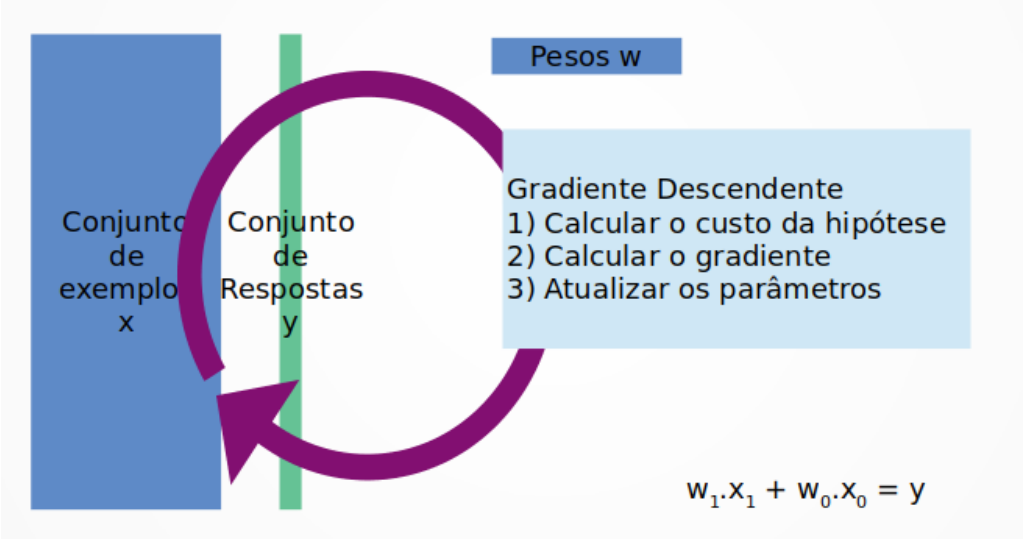

Figure 1. Gradiente descendente aplicado à regressão linear

1.2.1. Gradiente descendente em lotes

O gradiente descendente em lote (batch gradient descent) divide o conjunto de exemplos em várias partes e calcula o gradiente para cada lote progressivamente em vez de calcular todo o conjunto de uma só vez. Diminuir a quantidade de exemplos no lote pode viabilizar o processamento quando o conjunto de dados inteiro é muito grande para a memória.

1.2.2. Gradiente descendente estocástico

O gradiente descendente estocástico (stochastic gradient descent ou SGD) calcula o gradiente para cada um dos exemplos. Embora o comportamento seja bastante errático, ora aumentando ora diminuindo o valor dos parâmetros, é um método bem eficiente e bastante utilizado pois rapidamente fornece estimativas desde a primeira iteração.

1.2.3. Gradiente descendente em mini-lotes

É semalhante ao gradiente em lotes, mas trabalha com lotes bem pequenos. É uma opção intermediária entre o primeiro e o SGD, conjugando o melhor das duas abordagens. É o mais indicado para a maioria dos casos.

1.3. Paralelização do Gradiente Descendente

A paralelização do gradiente descendente pode ser síncrona ou assíncrona.

Na paralelização síncrona, os diversos workers trabalham em paralelo, sobre subconjuntos dos dados de treinamento, de forma semelhante ao gradiente descendente em lotes, porém compartilhando entre si os resultados obtidos. A cada nova epoch, os n processos são disparados, cada um processando uma parte dos dados. Ao final, todos enviam para todos o resultado obtido e iniciam nova rodada de treinamento.

O ponto fraco da paralelização síncrona é que caso a comunicação entre os processos seja interrompida ou um dos workers apresente falha, todo o processamento falha.

Na paralelização assíncrona, cada worker processa, a qualquer momento, um pedaço dos dados disponíveis e anota o resultado obtido em uma área compartilhada da memória, caso o valor por ela calculado tenha sido menor que o valor armazenado. Ao final dos pedaços, essa área estará preenchida com o menor valor. Esse algoritmo é conhecido como Hogwild! [Recht et al. 2011]

Como cada processo funciona de forma independente, a queda de comunicação ou falha de um deles não compromete o sucesso do processamento do algoritmo.

O Artigo [Chahal et al. 2018], ainda não publicado, discorre sobre as principais características e obstáculos enfrentados no processamento síncrono ou assíncrono do gradiente descendente.

No presente trabalho são testadas diversas abordagens de implementação do gradiente descendente paralelo síncrono e assíncrono.

2. Desenvolvimento

Para melhor visualização dos resultados, utilizamos o Jupyter Notebook e o arquivo GDParalelo.ipynb, onde apresentamos os diversos gráficos comparativos, e o arquivo utils.py contendo as funções que desenvolvemos para o processamento paralelo.

Parte das funções utilizadas, especialmente aquelas que simulam o processamento paralelo para fim de comparação, foram aproveitadas do código desenvolvido por Angad Gill e disponível em seu repositório Github, no endereço https://github.com/angadgill/Parallel-SGD.

2.1. Ambiente

A linguagem de desenvolvimento escolhida foi Python, na versão 3.6. Além das bibliotecas básicas, foram utilizadas:

- numpy
- matplotlib
- scikit-learn
- pymp
- pandas

Para reproduzir os resultados deste trabalho basta instalar o Anaconda, instalar as bibliotecas acima, manter os arquivos fornecidos em anexo na mesma pasta e acessar o notebook GDParalelo.ypnb através do Jupyter (eventualmente, será necessário mudar o caminho para o dataset no notebook).

2.2. Preparação de dados

Os dados utilizados nos experimentos são do dataset Diamonds, que pode ser obtido em https://www.kaggle.com/shivam2503/diamonds.

As variáveis categóricas alfanuméricas do dataset original foram convertidas em classes numéricas apropriadas para os testes através do script data_preparation.py, que está incluído entre os arquivos do projeto apenas para demonstrar como foram processados os dados originais.

O arquivo diamond_prices.csv já contém dados preparados para uso.

2.3. Experimentações

Neste trabalho foram implentadas 3 versões do gradiente descendente:

- Sequencial: processos executados sequencialmente, com compartilhamento de pesos incidental ou ao final.
- Paralelo síncrono: processos executados simultaneamente, com compartilhamento de pesos incidental ou ao final.
- Paralelo assíncrono: Processos executados simultaneamente, com atualização aleatória de pesos.

Os testes consistiram em comparar essas diferentes implementações em diversos cenários, com diferentes quantidades de workers.

Em todos os cenários o gradiente descendente é executado por 500 ciclos, refinando os valores dos pesos continuamente, mesmo que aqueles que apresentam o menor custo sejam rapidamente obtidos.

A quantidade de exemplos no dataset original (53 mil) foi reduzida para apenas 800 de modo a aumentar a dificuldade e fazer com que o gradiente precisasse de mais ciclos para convergir.

2.4. Cenários de testes

2.4.1. Cenário 1: Divisão do dataset em partes iguais

Neste cenário o dataset é dividido em partes iguais e cada uma é atribuída a um worker. A cada ciclo de treinamento, cada worker recalcula os pesos usando um mini-lote retirado de sua fração.

Apenas ao final do processamento de todos os workers é que calcula-se a média e obtém-se o resultado final.

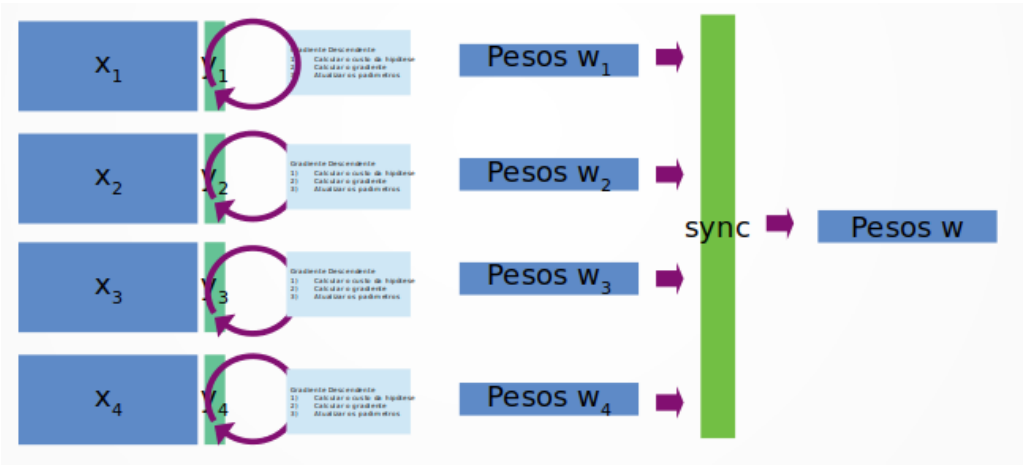

Figure 2. Cenário 1

Nota: Nas imagens, os workers são representados pelo retângulo em azul-claro. os w_n representam os pesos calculados por cada worker.

2.4.2. Cenário 2: Dataset integral

Neste cenário, todos os workers têm acesso a todo o conjunto de dados. Os pesos são finais são calculados uma única vez quando todos eles terminam o processamento.

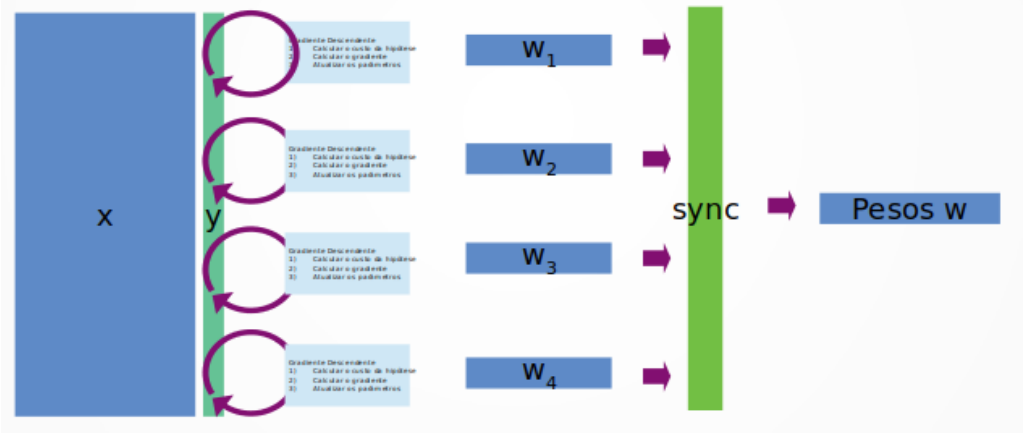

Figure 3. Cenário 2

2.4.3. Cenário 3: Divisão proporcional do dataset

Nesse cenário, grupos de workers recebem uma fração não contínua do dataset (com overlapping), de forma que cada um opere com um subconjunto aleatório dos dados.

Uma única sincronização é feita ao final do processamento de todos os workers.

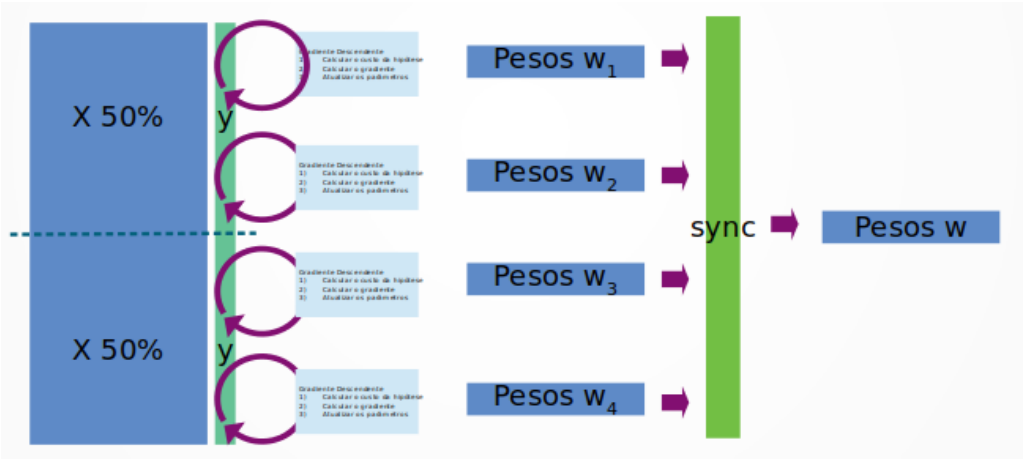

Figure 4. Cenário 3

2.4.4. Cenário 4: Dataset integral, com sincronizações intermediárias

Nesse cenário todo o dataset está disponível para todos os workers, mas os subconjuntos de dados acessados por cada um deles são aleatórios.

Também são introduzidas sincronizaços parciais dos pesos de cada worker, de modo que eles possam se beneficiar do progresso feito pelos demais processos antes de chegar ao final do processamento.

É nesse cenário que, pela primeira vez, ocorre a comunicação entre os processos paralelos durante o cálculo do gradiente descentente.

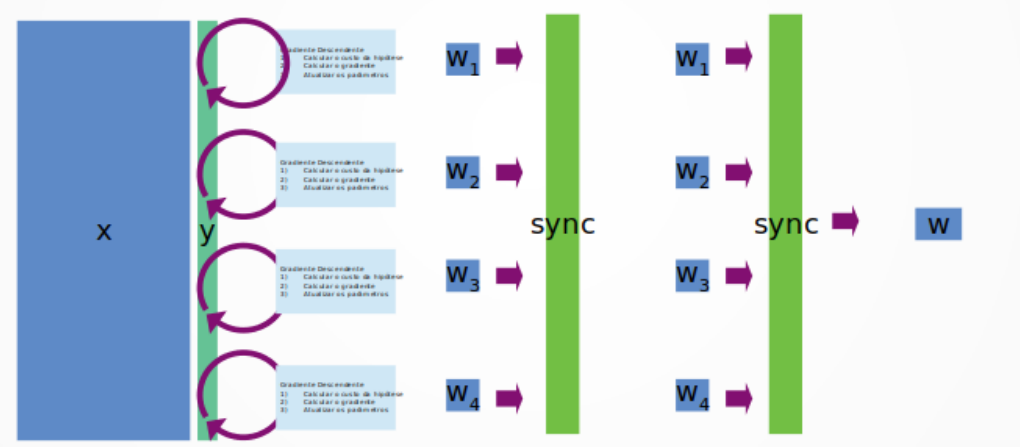

Figure 5. Cenário 4

2.4.5. Cenário 5: Processamento paralelo assíncrono

No cenário 5, implementamos o algoritmo de processamento assíncrono Hogwild!, onde qualquer worker acessa qualquer parte do conjunto de dados de treinamento e atualiza a qualquer momento o conjunto compartilhado de pesos.

Durate os testes, comparamos o hogwild com os outros tipos de processamento paralelo síncrono, com resultados expressivamente superiores.

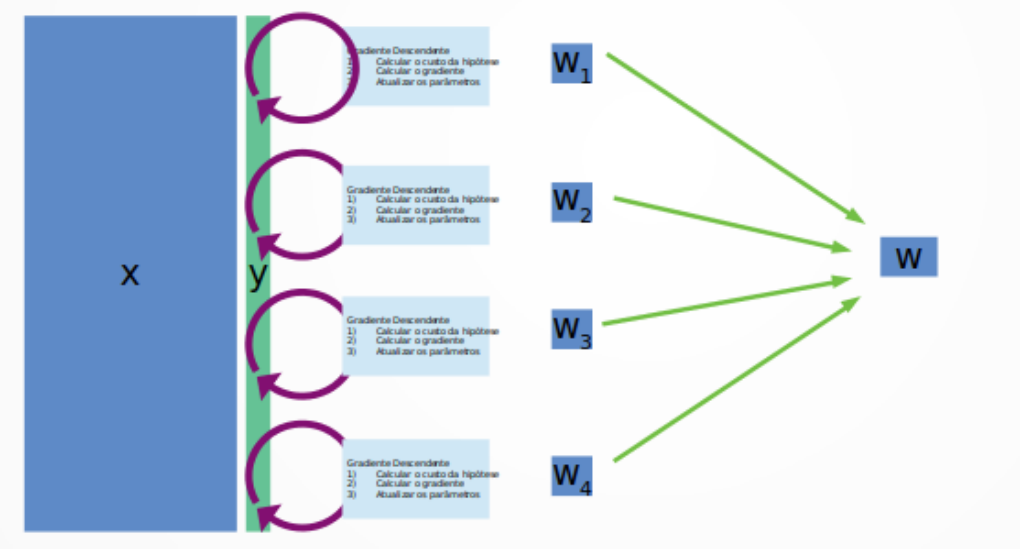

Figure 6. Cenário 5

3. Resultados

Os resultados dos diversos testes executados estão à disposição no material que acompanha este relatório e podem ser reproduzidos utilizando o código anexo.

Conseguimos demonstrar empiricamente que não apenas é possível, como também é recomendável o processamento paralelo do algoritmo do gradiente descendente.

Esta seção apresenta nossas conclusões.

3.1. Conclusões

3.1.1. Paralelismo sem sincronização não apresenta ganhos de performance

Quando são fornecidos os mesmos dados aos processos, mas não há comunicação entre processos durante o processamento do algoritmo, a paralelização não apresenta melhora no desempenho.

Entretanto, nos casos em que a regressão linear apresentar múltiplas mínimas locais, a paralelização pode aumentar as chances de o algoritmo encontrar a mínima global ao final do processamento.

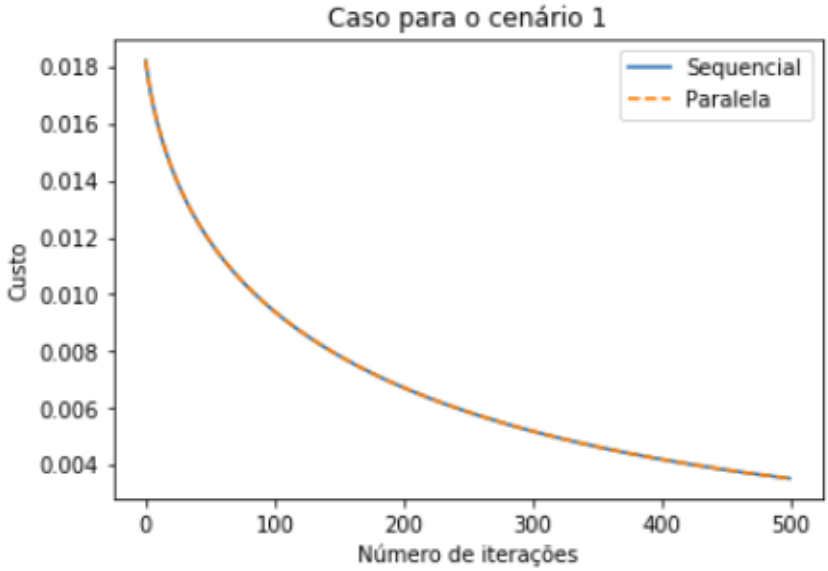

Figure 7. Conclusões

3.1.2. Minimização com ARGMIN é superior à com AVERAGE

O processamento paralelo síncrono converge mais rapidamente para a mínima global quando nas barreiras de sincronização é utilizada a função argmin em vez da média.

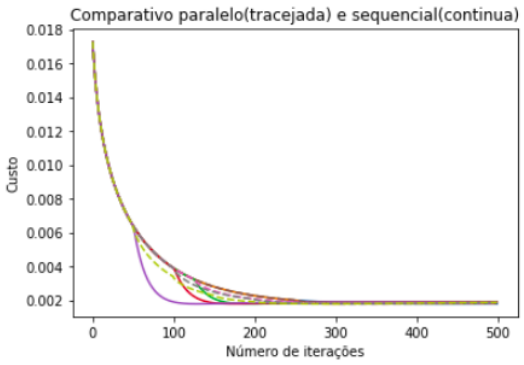

Figure 8. AVERAGE

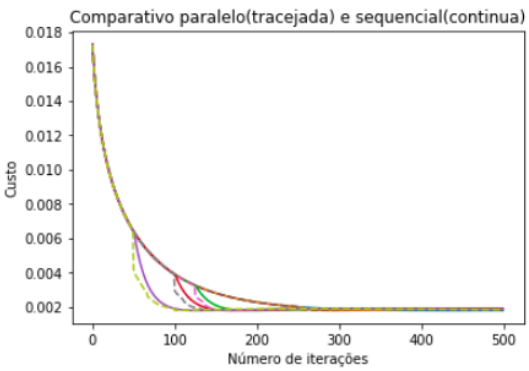

Figure 9. ARGMIN

3.1.3. Paralelização assíncrona é superior

No processamento assíncrono, quanto mais workers são utilizados, mais cedo se chega ao resultado. Ao mesmo tempo, a aleatoriedade do algoritmo diminui a possiblidade de o gradiente descendente se encerrar em uma minima local.

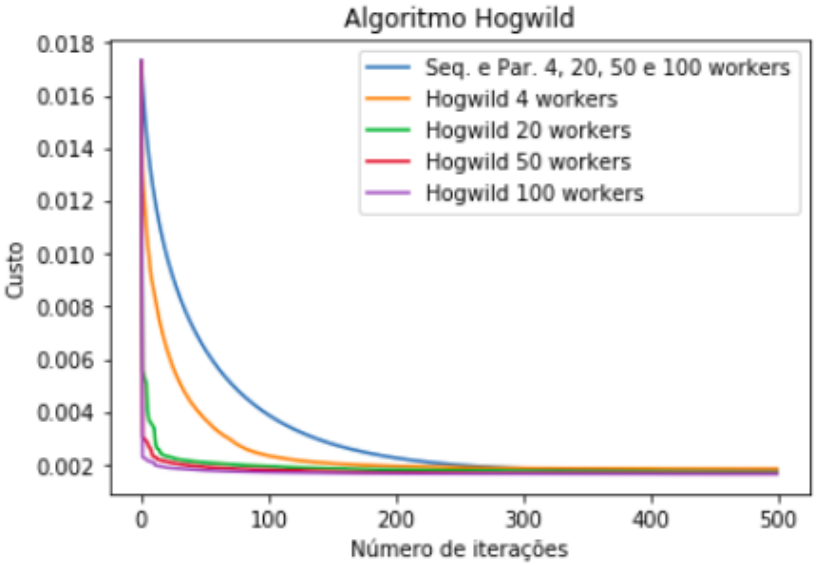

Figure 10. Aumento progressivo de workers paralelos

References

Chahal, K., Grover, M. S., and Dey, K. (2018). A hitchhiker's guide on distributed training of deep neural networks.

Recht, B., Re, C., Wright, S., and Niu, F. (2011). Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In Shawe-Taylor, J., Zemel, R. S., Bartlett, P. L., Pereira, F., and Weinberger, K. Q., editors, Advances in Neural Information Processing Systems 24, pages 693–701. Curran Associates, Inc.