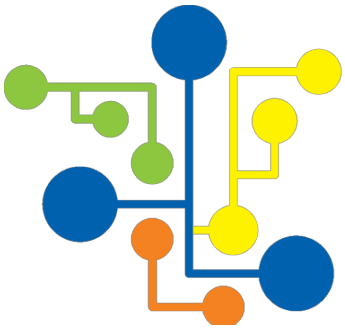


Data Science Academy

www.datascienceacademy.com.br

Matemática Para Machine Learning

Otimização Adam

A Otimização Adam incorpora todos os recursos interessantes da Otimização RMSProp e Gradiente com o momentum.

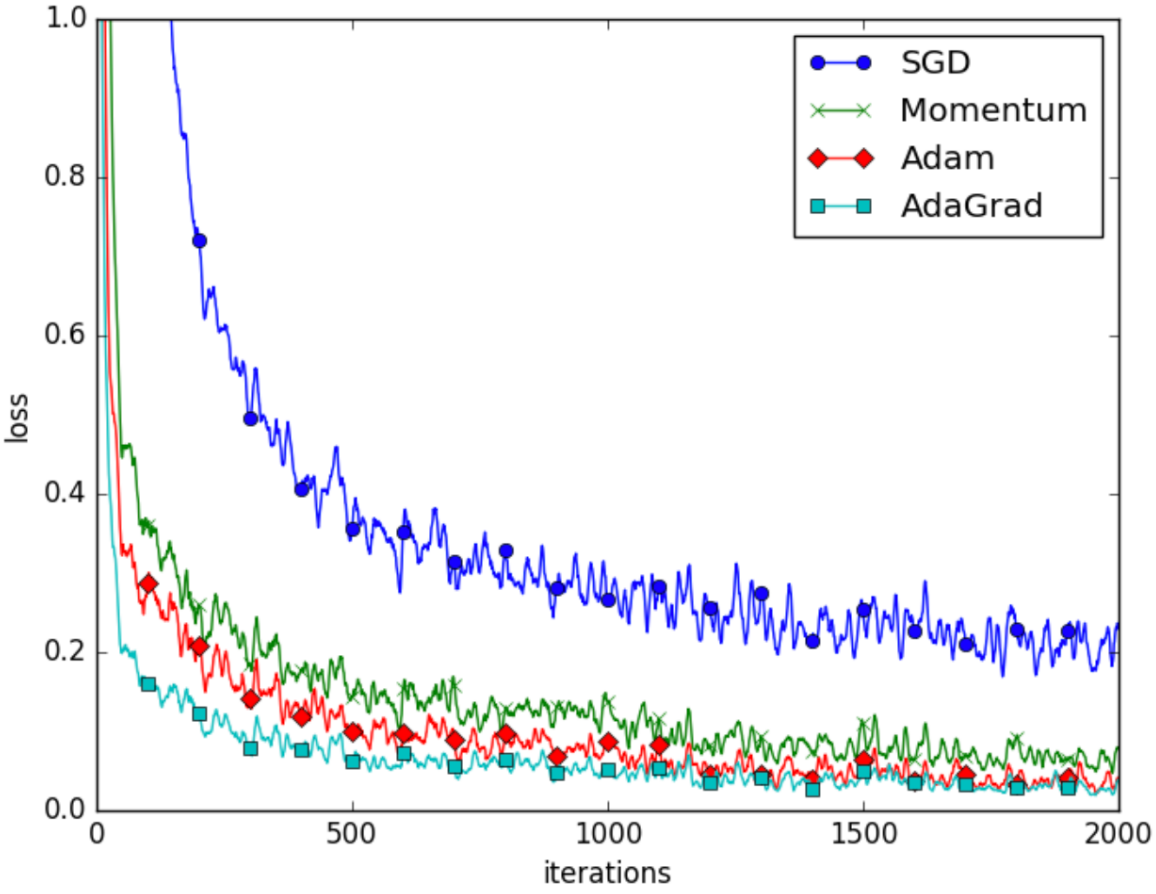
Especificamente, esse algoritmo calcula uma média móvel exponencial de gradientes e os gradientes quadrados, enquanto os parâmetros beta_1 e beta_2 controlam as taxas de decaimento dessas médias móveis.

```
first_moment = 0
second_moment = 0
for step in range(iterations_count):
    param_gradients = evaluate_gradients(loss_function,
                                         data,
                                         params)
    first_moment = beta_1 * first_moment + (1 - beta_1) *
                                                    param_gradients
    second_moment = beta_2 * second_moment + (1 - beta_2) *
                                  param_gradients * param_gradients
    params -= learning_rate * first_moment/(np.sqrt(second_moment) +
                                                            1e-8)
```

Observe que inicializamos second_moment com zero. Então, no começo, second_moment seria calculado como um lugar muito próximo de zero. Consequentemente, estamos atualizando os parâmetros dividindo com um número muito pequeno e, portanto, fazendo grandes atualizações no parâmetro. Isso significa inicialmente que o algoritmo faria passos maiores. Para corrigir, criamos uma estimativa imparcial desses primeiro e segundo momento, incorporando o passo atual. E então atualizamos os parâmetros com base nessas estimativas imparciais em vez de primeiro e segundo momentos. Matematicamente, em Python, seria assim:

```
first_moment = 0
second_moment = 0
for step in range(iterations_count):
    param_gradients = evaluate_gradients(loss_function,
                                         data,
                                         params)
    first_moment = beta_1 * first_moment + (1 - beta_1) *
                                                     param_gradients
    second_moment = beta_2 * second_moment + (1 - beta_2) *
                                  param_gradients * param_gradients
    first_bias_correction = first_moment/(1 - beta_1 ** step)
    second_bias_correction = second_moment/(1 - beta_2 ** step)
    params -= learning_rate * first_bias_correction/
                           (np.sqrt(second_bias_correction) + 1e-8)
```

A figura abaixo demonstra o desempenho de cada um dos algoritmos de otimização à medida que as iterações passam. Claramente, adicionar *momentum* aumenta a precisão. Na prática, no entanto, Adam é conhecido por executar muito bem com grandes conjuntos de dados e recursos complexos, sendo em geral a melhor opção.



Referências:

http://cs229.stanford.edu/notes/cs229-notes1.pdf

https://stanford.edu/~rezab/classes/cme323/S15/notes/lec11.pdf

https://arxiv.org/abs/1609.04747

http://ruder.io/optimizing-gradient-descent/