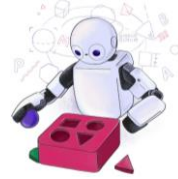


T319 - Introdução ao
Aprendizado de Máquina:
Regressão Linear (Parte II)



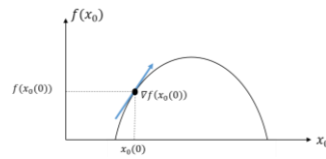
Inatel

Felipe Augusto Pereira de Figueiredo
felipe.figueiredo@inatel.br

Recapitulando

- Vimos a motivação por trás da regressão: encontrar curvas que nos ajudem a prever valores.
- Definimos o problema matematicamente.
- Vimos como resolver o problema da regressão, i.e., encontrar os pesos do modelo, através da equação normal.
- Aprendemos o que é uma superfície de erro.
- Discutimos algumas desvantagens da equação normal e apresentamos uma solução para essas desvantagens, a qual discutiremos a seguir.

Vetor Gradiente



- Vocês se lembram das aulas de cálculo vetorial, onde vocês aprenderam sobre o **vetor gradiente**?
 - Vetor gradiente é um vetor que indica a direção na qual, por deslocamento a partir de um ponto especificado, obtém-se o maior incremento possível no valor de uma função, f .
- O **vetor gradiente** de uma função, $f(x_0, x_1, \dots, x_K)$, em relação aos seus argumentos $x_k, k = 0, \dots, K$, é definido por

$$\nabla f(x_0, x_1, \dots, x_K) = \left[\frac{\partial f(x_0, x_1, \dots, x_K)}{\partial x_0} \quad \frac{\partial f(x_0, x_1, \dots, x_K)}{\partial x_1} \quad \dots \quad \frac{\partial f(x_0, x_1, \dots, x_K)}{\partial x_K} \right]^T,$$

onde $\nabla f(x_0, x_1, \dots, x_K)$ é o vetor que aponta a direção em que a função, $f(x_0, x_1, \dots, x_K)$, tem a taxa de crescimento mais rápida.

- Notem, que cada elemento do vetor gradiente aponta para a direção de máxima variação em relação àquele argumento da função.
- Se imaginem parados no ponto $x_0(0), x_1(0), \dots, x_K(0)$ no domínio de f , o vetor $\nabla f(x_0(0), x_1(0), \dots, x_K(0))$ diz em que direção vocês devem caminhar para aumentar o valor de f mais rapidamente.

A coisa mais importante para nos lembrarmos sobre o vetor gradiente é que ele sempre apontará para a direção onde a subida da função é mais íngreme.

$\nabla f \Rightarrow$ Nabla f

O gradiente pode ser interpretado como a "direção em que uma função tem taxa de aumento mais rápido".

O valor do gradiente em um ponto é um vetor tangente ao ponto. Valores positivos indicam que o aumenta mais rápido esta à frente, já valores negativos indicam que a taxa de aumento mais rápida está para trás. O valor zero indica que estamos sobre o máximo.

Imagine você parado no ponto x de uma função, f , o vetor ∇f diz em qual direção você deve caminhar para aumentar o valor da função f mais rapidamente.

Se você seguir na direção do gradiente, você chegará ao máximo da função.

Note que cada elemento do vetor gradiente aponta para a direção de máxima variação em relação à aquele argumento/parâmetro.

Referências relacionadas com a obtenção das derivadas parciais

<https://mccormickml.com/2014/03/04/gradient-descent-derivation/>

<https://towardsdatascience.com/understanding-the-mathematics-behind-gradient-descent-dde5dc9be06e>

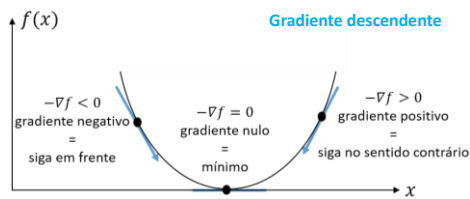
Gradiente Ascendente



- O valor do vetor gradiente em um ponto é um vetor tangente àquele ponto, onde um elemento do vetor com valor:
 - + significa que o ponto de máximo está à frente.
 - - significa que o ponto de máximo está atrás.
 - 0 significa que ponto de máximo foi encontrado.
- Portanto, o **vetor gradiente** nos permite encontrar o ponto de **máximo** da função, $f(x_0, x_1, \dots, x_K)$.
 - Seguindo na direção do vetor gradiente, chegamos ao ponto de máximo da função.
- Assim, um algoritmo de otimização **iterativo** que siga a direção dada pelo vetor gradiente para encontrar o **ponto de máximo** de $f(x_0, x_1, \dots, x_K)$ é conhecido como **gradiente ascendente**.

Gradiente Descendente

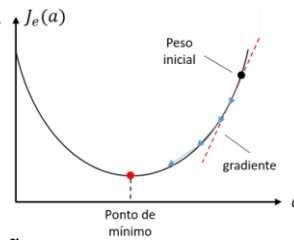
- Mas e se formos na direção contrária a da taxa de crescimento, dada pelo vetor gradiente, $\nabla f(x_0, x_1, \dots, x_K)$, ou seja $-\nabla f(x_0, x_1, \dots, x_K)$?
 - Nesta caso, iremos na direção de **decremento** mais rápido da função, $f(x_0, x_1, \dots, x_K)$.
- Portanto, um algoritmo de otimização **iterativo** que siga a direção contrária a dada pelo vetor gradiente para encontrar o **ponto de mínimo** de $f(x_0, x_1, \dots, x_K)$ é conhecido como **gradiente descendente**.



um algoritmo de otimização **iterativo** que use o vetor gradiente para encontrar o ponto de máximo de $f(x_0, x_1, \dots, x_K)$ é conhecido como **gradiente ascendente**.

Gradiente Descendente

- Algoritmo de otimização **iterativo** e **genérico** capaz de encontrar soluções ótimas para uma ampla gama de problemas.
- É utilizado em vários problemas de aprendizado de máquina.
- Escalona melhor do que o método de **equação normal** para grandes conjuntos de dados.
- É de fácil implementação.
- Não é necessário se preocupar com matrizes mal-condicionadas (determinante próximo de 0, i.e., quase **singulares**).
- O único requisito é que a **função de erro** seja **diferenciável**.
- Quando aplicado a problemas de **regressão**, a ideia geral é ajustar os pesos, **α** , iterativamente, a fim de **minimizar a função de erro**, ou seja, encontrar seu **ponto de mínimo**.
- A seguir, veremos como aplicar o algoritmo do gradiente descendente ao problema da regressão linear.



O Gradiente Descendente (GD) é um algoritmo de otimização iterativo e genérico capaz de encontrar soluções ideais para uma ampla gama de problemas.

Métodos iterativos de otimização são usados toda a parte de Aprendizado de Máquina.

A ideia geral do GD é ajustar os parâmetros iterativamente, a fim de minimizar uma função de custo.

Para algoritmos que utilizam o Gradiente Descendente para otimizar os parâmetros do modelo, todas as funções devem ser diferenciáveis.

O Gradiente descendente é utilizado em vários problemas de aprendizado de máquina.

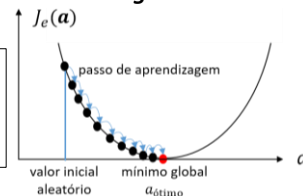
Não precisa se preocupar com matrizes mal-condicionadas, ou seja, com determinante próximo de zero.

O Algoritmo do Gradiente do Descendente (GD)

- O algoritmo inicializa os pesos, \mathbf{a} , em um ponto aleatório do **espaço de pesos** e então, os atualiza na direção oposta ao do **gradiente** até que algum critério de convergência seja atingido, indicando que um **mínimo local** ou o **global** da **função de erro** foi encontrado.

$\mathbf{a} \leftarrow$ inicializa em um ponto qualquer do espaço de parâmetros
loop até convergir **ou** atingir número máximo de épocas **do**
 for each a_i in \mathbf{a} **do**

$$a_i \leftarrow a_i - \alpha \frac{\partial J_e(\mathbf{a})}{\partial a_i}$$



onde α é a **taxa/passo de aprendizagem** e $\frac{\partial J_e(\mathbf{a})}{\partial a_i}$ é o gradiente da **função de erro** em relação ao parâmetro a_i .

- A **taxa de aprendizagem** dita o tamanho dos passos/deslocamento dado na direção oposta à do gradiente.

Inicializa as parâmetros/pesos, \mathbf{a} , em um ponto aleatório do espaço de parâmetros e então, atualiza os parâmetros na direção oposta ao do gradiente até que algum critério de convergência seja atingido, indicando que o mínimo global ou um mínimo local da função de erro/custo foi encontrado.

Taxa de aprendizado: tamanho dos passos/deslocamento dado na direção oposta ao gradiente.

Referências relacionadas com a obtenção das derivadas parciais

<https://mccormickml.com/2014/03/04/gradient-descent-derivation/>

<https://towardsdatascience.com/understanding-the-mathematics-behind-gradient-descent-dde5dc9be06e>

O Algoritmo do Gradiente do Descendente (GD)

- O **passo de aprendizagem**, α , pode ser constante ou pode decair com o tempo à medida que o processo de aprendizado prossegue.
- **OBS.:** Os parâmetros, \mathbf{a} , **devem ser atualizados simultaneamente**, caso contrário o algoritmo apresentará comportamento desconhecido.
- O pseudo-algoritmo abaixo apresenta a atualização simultânea de todos os **pesos**.

```

 $\mathbf{a} \leftarrow$  inicializa em um ponto qualquer do espaço de parâmetros
loop até convergir ou atingir número máximo de épocas do
     $\mathbf{a} \leftarrow \mathbf{a} - \alpha \nabla J_e(\mathbf{a})$ 
    
```

onde \mathbf{a} é o vetor com os **pesos** e $\nabla J_e(\mathbf{a}) = \left[\frac{\partial J_e(\mathbf{a})}{\partial a_1} \quad \dots \quad \frac{\partial J_e(\mathbf{a})}{\partial a_K} \right]^T$ é o **vetor gradiente**, o qual contém o gradiente com relação a todos os **pesos**.

- Na sequência, veremos como encontrar o **vetor gradiente** da função de erro e implementar o algoritmo do gradiente descendente.

Inicializa parâmetros, \mathbf{a} , em um ponto aleatório do espaço de parâmetros e então, atualiza os parâmetros na direção oposta ao do gradiente até que algum critério de convergência seja atingido, indicando que o mínimo global da função de erro/custo foi encontrado.

Taxa de aprendizado: tamanho dos passos/deslocamento dado na direção oposta ao gradiente.

Referências relacionadas com a obtenção das derivadas parciais

<https://mccormickml.com/2014/03/04/gradient-descent-derivation/>

<https://towardsdatascience.com/understanding-the-mathematics-behind-gradient-descent-dde5dc9be06e>

Exemplo #1

[Exemplo 1: linear regression with gradient descent exemplo1.ipynb](#)

Neste exemplo, usaremos uma **função hipótese** com 2 pesos, a_1 e a_2 , sendo $a_0 = 0$

$$\hat{y}(n) = h(x(n)) = a_1 x_1(n) + a_2 x_2(n).$$

A função de erro é dada por

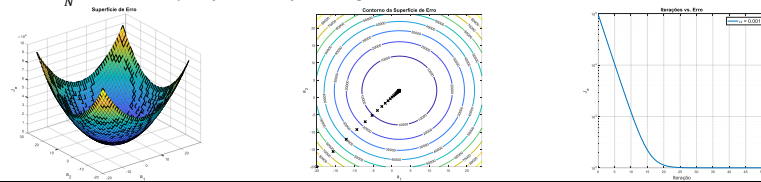
$$J_e(\mathbf{a}) = \frac{1}{N} \sum_{n=0}^{N-1} [y(n) - (a_1 x_1(n) + a_2 x_2(n))]^2.$$

E atualização dos parâmetros $a_k, k = 1$ e 2 dada por

$$\frac{\partial J_e(\mathbf{a})}{\partial a_k} = -\frac{2}{N} \sum_{n=0}^{N-1} [y(n) - (a_1 x_1(n) + a_2 x_2(n))] x_k(n), \quad k = 1, 2,$$

$$a_k = a_k - \alpha \frac{\partial J_e(\mathbf{a})}{\partial a_k} \therefore a_k = a_k + \alpha \sum_{n=0}^{N-1} [y(n) - (a_1 x_1(n) + a_2 x_2(n))] x_k(n), \quad k = 1, 2.$$

onde o termo $\frac{2}{N}$ foi absorvido pelo **passo de aprendizagem**, α .



Exemplo # 1:

https://mybinder.org/v2/gh/zz4fap/t319_aprendizado_de_maquina/main?filepath=notebooks%2Fregression%2Flinear_regression_with_gradient_descent_exemplo1.ipynb

Figuras: [linear_regression_with_gradient_descente_exemplo2.m](#)

Nesse exemplo, o vetor \mathbf{a} inicial é inicializado com os valores $[-20; -20]$ e vemos que o algoritmo caminha progressivamente em direção ao mínimo global mostrado pelo asterisco em vermelho (esse ponto foi calculado com o método da equação normal). O passo de aprendizado é feito igual a 0.001 e como podemos ver o algoritmo converge à partir da iteração número 25.

A figura do meio mostra a trajetória realizada pelo algoritmo até a convergência.

Vejam que o algoritmo converge lentamente e portanto, é possível aumentar o passo de aprendizagem.

O passo de aprendizado, α , pode ser um valor constante ou pode decair com o tempo à medida que o processo de aprendizado prossegue.

Referências relacionadas com a obtenção das derivadas parciais

<https://mccormickml.com/2014/03/04/gradient-descent-derivation/>

<https://towardsdatascience.com/understanding-the-mathematics-behind-gradient-descent-dde5dc9be06e>

Exemplo #2

[Exemplo: linear_regression_with_gradient_descent_exemplo2.ipynb](#)

Agora consideramos uma **função hipótese** com os pesos, a_0 e a_1 ,

$$\hat{y}(n) = h(x(n)) = a_0 + a_1 x_1(n).$$

A **função de erro** é dada por

$$J_e(\mathbf{a}) = \frac{1}{N} \sum_{n=0}^{N-1} [y(n) - (a_0 + a_1 x_1(n))]^2.$$

E a atualização dos pesos a_k , $k = 0$ e 1 é dada por

$$\frac{\partial J_e(\mathbf{a})}{\partial a_k} = -\frac{2}{N} \sum_{n=0}^{N-1} [y(n) - (a_0 + a_1 x_1(n))] x_k(n), \quad k = 0, 1,$$

$$a_k = a_k - \alpha \frac{\partial J_e(\mathbf{a})}{\partial a_k} \therefore a_k = a_k + \alpha \sum_{n=0}^{N-1} [y(n) - (a_0 + a_1 x_1(n))] x_k(n), \quad k = 0, 1,$$

onde $x_0(n) = 1 \forall n$.

OBS.1: Temos o termo de bias nesta função hipótese, portanto, não se esqueçam da coluna de '1's na implementação do código.

OBS.2: Para executar este exemplo, é necessário instalar a biblioteca ffmpeg com o comando: `conda install ffmpeg`

Exemplo:

https://mybinder.org/v2/gh/zz4fap/t319_aprendizado_de_maquina/main?filepath=notebooks%2Fregression%2Flinear_regression_with_gradient_descent_exemplo2.ipynb

Para executar esse exemplo é necessário instalar a biblioteca ffmpeg com o comando: `conda install ffmpeg`

Referências relacionadas com a obtenção das derivadas parciais

<https://mccormickml.com/2014/03/04/gradient-descent-derivation/>

<https://towardsdatascience.com/understanding-the-mathematics-behind-gradient-descent-dde5dc9be06e>

Generalizando a equação de atualização

- Baseado no que vimos nos exemplos anteriores, podemos generalizar a **equação de atualização do pesos** da seguinte forma:

$$\frac{\partial J_e(\mathbf{a})}{\partial a_k} = -\frac{2}{N} \sum_{n=0}^{N-1} [y(n) - \hat{y}(n)] x_k(n), \forall k,$$

$$a_k = a_k - \alpha \frac{\partial J_e(\mathbf{a})}{\partial a_k}$$

$$a_k = a_k + \alpha \sum_{n=0}^{N-1} [y(n) - \hat{y}(n)] x_k(n), \forall k.$$

- Essa equação pode ser aplicada a qualquer problema de regressão linear.
- Apenas não se esqueçam de que quando $k = 0$, $x_0(n) = 1, \forall n$.

Versões do Gradiente Descendente

Existem 3 diferentes versões para a implementação do algoritmo do Gradiente Descendente: Batelada, Estocástico e Mini-Batch.

- **Batelada (do inglês *batch*)**: como já vimos, a cada **época** do algoritmo, **todos** os exemplos de treinamento são considerados no processo de treinamento do modelo. Esta foi versão foi utilizada nos exemplos 1 e 2.

$$a_k = a_k + \alpha \sum_{n=0}^{N-1} [y(n) - (a_1 x_1(n) + a_2 x_2(n))] x_k(n), \quad k = 1, \dots, K$$

- Utilizado quando se possui previamente todos os atributos e rótulos de treinamento, ou seja, o conjunto de treinamento.
- **Convergência garantida**, dado que o passo de aprendizagem não seja grande demais e se espere o tempo necessário para a convergência.
- **Convergência pode ser bem lenta**, dado que o modelo é apresentado a todos os exemplos a cada época, ou seja, a atualização dos pesos só é realizada após o processamento de todo o conjunto de treinamento.

Batch: usa todos os exemplos de treinamento a cada iteração. Como resultado, é muito lento em conjuntos de treinamento muito grandes.

A versão **online** seleciona apenas uma instância aleatória no conjunto de treinamento a cada etapa e calcula os gradientes com base apenas nessa única instância. Obviamente, isso torna o algoritmo muito mais rápido, pois possui muito poucos dados para manipular a cada iteração. Por outro lado, devido à sua natureza estocástica (ou seja, aleatória), esse algoritmo é muito menos regular do que a descida do gradiente em lote: em vez de diminuir suavemente até atingir o mínimo, a função de custo irá saltar para cima e para baixo, diminuindo apenas em média .

Mini-batch: em cada iteração, em vez de calcular os gradientes com base no conjunto de treinamento completo (como no Batch) ou com base em apenas uma instância (como no GD estocástico), o mini-batch GD calcula os gradientes em pequenos conjuntos aleatórios de instâncias chamados mini-batches.

Como todas as coisas neste mundo, todas as três versões do GD que vimos têm suas vantagens e desvantagens. Não é como se uma versão fosse usada com mais frequência do que as outras. Cada versão é usada igualmente, dependendo da situação e do contexto do problema.

Época

Uma época é quando todo o conjunto de dados (exemplos) de treinamento é utilizado no treinamento do modelo.

Iterações

Iteração corresponde a um batch apresentado ao modelo.

Conta o número de batches necessários para concluir uma época, caso cada batch seja menor do que o conjunto de treinamento.

Tamanho do batch

Número total de exemplos de treinamento presentes em um único batch que será utilizado durante uma iteração de treinamento.

Se um batch conter todos os exemplos de treinamento então cada iteração é igual a uma época.

Versões do Gradiente Descendente

- **Gradiente Descendente Estocástico (GDE)**: também conhecido como **online** ou **incremental** (exemplo-a-exemplo). Nessa versão, os **pesos** do modelo são atualizados a cada novo exemplo de treinamento.

$$a_k = a_k + \alpha [y(n) - (a_1 x_1(n) + a_2 x_2(n))] x_k(n), \quad k = 1, \dots, K$$

- **Aproxima o gradiente** através de uma **estimativa estocástica**: aproximação através do gradiente calculado com um único exemplo de treinamento.
- Pode ser utilizado quando os **atributos** e **rótulos** são obtidos sequencialmente, ou seja, de forma online, exemplo a exemplo.
- Ou quando o conjunto de treinamento é muito grande. Nesse caso, escolhe-se **aleatoriamente** um par atributos/rótulo a cada **iteração** (i.e., atualização dos pesos).
- **Convergência mais rápida**.
- **Porém, ela não é garantida** com um passo de aprendizagem fixo. O algoritmo pode oscilar em torno do mínimo sem nunca convergir para os valores ótimos.
- O uso de esquemas para variação do passo de aprendizagem garantem a convergência.

Métodos de **aproximação estocástica** são uma família de métodos iterativos normalmente usados para problemas de procura de raízes ou para problemas de otimização.

Pode ser considerada como uma aproximação estocástica da otimização do gradiente descendente, uma vez que substitui o valor do gradiente real (calculado a partir de todo o conjunto de dados) por uma estimativa do mesmo (calculado a partir de um subconjunto de exemplos selecionado aleatoriamente).

O insight por trás do gradiente descendente estocástico é que o gradiente é uma esperança, ou seja, uma média. Portanto, a esperança pode ser estimada aproximadamente usando um pequeno conjunto de exemplos.

Como todas as coisas neste mundo, todas as três versões do GD que vimos têm suas vantagens e desvantagens. Não é como se uma versão fosse usada com mais frequência do que as outras. Cada versão é usada igualmente, dependendo da situação e do contexto do problema.

Versões do Gradiente Descendente

- **Mini-batch:** é um meio-termo entre as duas versões anteriores. O conjunto de treinamento é dividido em vários subconjuntos (mini-batches) com elementos (i.e., pares atributos/rótulo) aleatoriamente retirados do conjunto de treinamento, onde os pesos do modelo são ajustados a cada mini-batch.

$$a_k = a_k + \alpha \sum_{n=0}^{MB-1} [y(n) - (a_1 x_1(n) + a_2 x_2(n))] x_k(n), \quad k = 1, \dots, K$$

onde MB é o tamanho do mini-batch.

- Pode ser visto como uma generalização das 2 versões anteriores:
 - Caso $MB = N$, então se torna o GD em batelada.
 - Caso $MB = 1$, então se torna o GD estocástico.
- Tem convergência mais rápida do que o GD em batelada, mas mais lenta do que o GD estocástico.
- Convergência depende do tamanho do mini-batch, quanto maior o tamanho de MB maior a chance de convergência.
- Pode usar esquemas de variação do passo de aprendizagem para melhorar a convergência.

Batch: usa todos os exemplos de treinamento a cada iteração. Como resultado, é muito lento em conjuntos de treinamento muito grandes.

A versão **online** seleciona apenas uma instância aleatória no conjunto de treinamento a cada etapa e calcula os gradientes com base apenas nessa única instância. Obviamente, isso torna o algoritmo muito mais rápido, pois possui muito poucos dados para manipular a cada iteração. Por outro lado, devido à sua natureza estocástica (ou seja, aleatória), esse algoritmo é muito menos regular do que a descida do gradiente em lote: em vez de diminuir suavemente até atingir o mínimo, a função de custo irá saltar para cima e para baixo, diminuindo apenas em média.

Mini-batch: em cada iteração, em vez de calcular os gradientes com base no conjunto de treinamento completo (como no Batch) ou com base em apenas uma instância (como no GD estocástico), o mini-batch GD calcula os gradientes em pequenos conjuntos aleatórios de instâncias chamados mini-batches.

O insight por trás do gradiente descendente estocástico é que o gradiente é uma esperança, ou seja, uma média. Portanto, a esperança pode ser estimada aproximadamente usando um pequeno conjunto de exemplos.

Como todas as coisas neste mundo, todas as três versões do GD que vimos têm suas vantagens e desvantagens. Não é como se uma versão fosse usada com mais frequência do que as outras. Cada versão é usada igualmente, dependendo da situação e do contexto do problema.

Implementação: GD em Batelada

[Exemplo: batch gradient descent with figures.ipynb](#)

```
import numpy as np

# Define the number of examples.
N = 1000

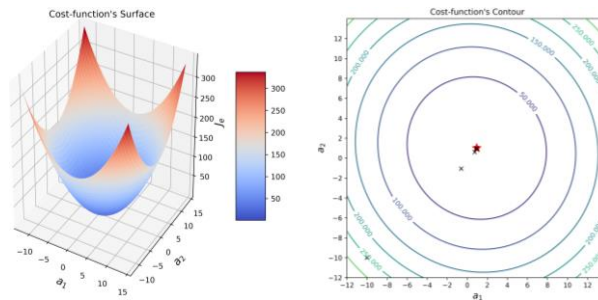
# Generate target function.
x1 = np.random.randn(N, 1)
x2 = np.random.randn(N, 1)
y = x1 + x2 + np.random.randn(N, 1)

# Concatenate both column vectors, x1 and x2.
X = np.c_[x1, x2]

# Constant learning rate.
alpha = 0.1
# Number of iterations.
n_iterations = 1000

# Random initialization.
a = np.random.randn(2, 1)

# Batch gradient-descent loop.
for iteration in range(n_iterations):
    gradients = -2/N * X.T.dot(y - X.dot(a))
    a = a - alpha * gradients
```



- Segue diretamente para o mínimo global.
- Atinge o mínimo em 4 épocas.
- Nesse caso específico, segue linha reta entre θ_0 e θ_1 pois a taxa de decrescimento da superfície de erro é igual para os dois parâmetros (contornos são circulares).
- Não fica “oscilando” em torno do mínimo após alcançá-lo.
- Algoritmo para no mínimo pois o vetor gradiente no ponto ótimo é praticamente nulo.

Exemplo:

[https://mybinder.org/v2/gh/zz4fap/t319_aprendizado_de_maquina/main?filepath=no tebooks%2Fregression%2Fgd_versions%2Fbatch_gradient_descent_with_figures.ipynb](https://mybinder.org/v2/gh/zz4fap/t319_aprendizado_de_maquina/main?filepath=no%20tebooks%2Fregression%2Fgd_versions%2Fbatch_gradient_descent_with_figures.ipynb)

Não fica “oscilando”, “dançando” ou “ricocheteando” ou “zig-zagueando” em torno do mínimo após chegar próximo dele.

Implementação: Gradiente Descendente Estocástico

```
import numpy as np

# Define the number of examples.
N = 1000

# Generate target function.
x1 = np.random.randn(N, 1)
x2 = np.random.randn(N, 1)
y = x1 + x2 + np.random.randn(N, 1)

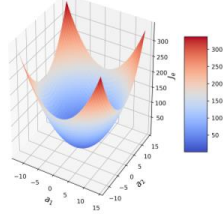
# Concatenate both column vectors, x1 and x2.
X = np.c_[x1, x2]

# Number of epochs.
n_epochs = 1
# Constant learning rate.
alpha = 0.1

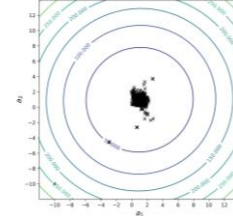
# Random initialization of parameters.
a = np.random.randn(2,1)

# Stochastic gradient-descent loop.
for epoch in range(n_epochs):
    for i in range(N):
        random_index = np.random.randint(N)
        xi = X[random_index:random_index+1]
        yi = y[random_index:random_index+1]
        gradients = -2*xi.T.dot(yi - xi.dot(a))
        a = a - alpha * gradients
```

Cost-function's Surface



Cost-function's Contour



Exemplo: stochastic gradient descent with figures.ipynb

- Devido à sua natureza estocástica, não apresenta um caminho regular/direto para o mínimo, mudando de direção várias vezes.
- Por aproximar o gradiente com apenas um exemplo, nem sempre estamos iremos na direção ideal, porque as derivadas parciais são "ruidosas".
- O algoritmo não diminui suavemente até atingir o mínimo, fica "oscilando" ou "ricocheteando" em torno dele.
- Quando o algoritmo para, os valores finais dos parâmetros são bons, mas não são ótimos.
- A convergência ocorre apenas na média.
- Tempo de treinamento é menor, nesse caso, com apenas uma época o algoritmo já se aproxima do ponto ótimo.
- Necessita de um esquema de ajuste do passo de aprendizagem, α , para ficar mais "comportado". Por exemplo, pode-se diminuir o valor do passo conforme o algoritmo caminha em direção ao mínimo (discutiremos isso a seguir).

Exemplo:

https://mybinder.org/v2/gh/zz4fap/t319_aprendizado_de_maquina/main?filepath=notebooks%2Fregression%2Fgd_versions%2Fstochastic_gradient_descent_with_figures.ipynb

Devido à sua natureza estocástica (ou seja, aleatória), esse algoritmo é muito menos regular do que o gradiente descendente em batelada: em vez de diminuir suavemente até atingir o mínimo, o gradiente da função de custo irá saltar para cima e para baixo, convergindo apenas na média. Com o passar do tempo, o algoritmo terminará muito próximo do mínimo, mas, quando chegar lá, continuará a ricocheteiar/oscilar, nunca convergindo (o gradiente estocástico nunca zera definitivamente). Portanto, quando o algoritmo para, os valores finais dos parâmetros são bons, mas não são ótimos.

Quando a função de custo é muito irregular, essa aleatoriedade do algoritmo pode realmente ajuda-lo a escapar de mínimos locais quando temos funções de custo não-convexas, de modo que o gradiente descendente estocástico tem uma chance maior de encontrar o mínimo global do que o gradiente descendente em batelada.

A aleatoriedade do algoritmo é uma faca de dois gumes, pois é boa para escapar de mínimos locais, mas é ruim pois significa que o algoritmo nunca irá se "acomodar" no mínimo global. Uma solução para esse dilema é reduzir gradualmente a taxa de aprendizagem. Os passos começam com grandes valores (o que ajuda a progredir/aprender rapidamente e a escapar de mínimos locais) e depois diminuem cada vez mais, permitindo que o algoritmo se estabilize no mínimo global.

Exemplo: stochastic_gradient_descent_with_figures.ipynb

Implementação: GD com Mini-Batch

```
import numpy as np

# Define the number of examples.
M = 1000
```

```
# Generate target function.
x1 = np.random.randn(M, 1)
x2 = np.random.randn(M, 1)
y = x1 + x2 + np.random.randn(M, 1)
```

```
# Concatenate both column vectors, x1 and x2.
X = np.c_[x1, x2]
```

```
# Constant learning rate.
alpha = 0.1
```

```
# Number of iterations.
n_iterations = 1000
```

```
# Random initialization.
a = np.random.randn(2, 1)
```

```
# Mini-batch size.
mb_size = 10
```

```
# Mini-batch gradient-descent loop.
```

```
for epoch in range(n_epochs):
    sdi = random.sample(range(0, N), N)
    for i in range(0, N/mb_size):
        bi = sdi[i*mb_size:mb_size*(i+1)]
```

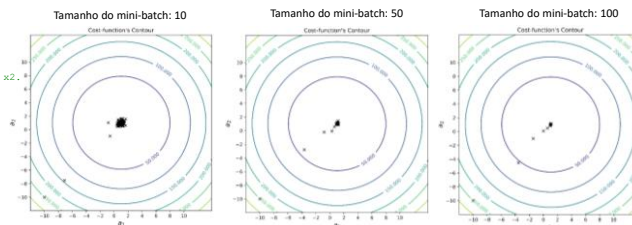
```
        xi = X[bi]
```

```
        yi = y_noisy[bi]
```

```
        gradients = -(2.0/mb_size)*xi.T.dot(yi - xi.dot(a))
```

```
        a = a - alpha*gradients
```

[Exemplo: mini_batch_gradient_descent_with_figures.ipynb](#)



- O progresso do algoritmo no espaço de parâmetros é menos irregular do que com o GD estocástico, especialmente com mini-batches grandes o suficiente.
- Como resultado, o mini-batch se aproxima mais do mínimo global do que o GDE.
- Tem comportamento mais próximo do GD em batelada.
- Oscilação em torno do mínimo diminui conforme o tamanho do mini-batch aumenta.
- Pode também ser usado com um esquema de variação do passo de aprendizagem.

Exemplo:

https://mybinder.org/v2/gh/zz4fap/t319_aprendizado_de_maquina/main?filepath=no%20tebooks%2Fregression%2Fgd_versions%2Fmini_batch_gradient_descent_with_figures.ipynb

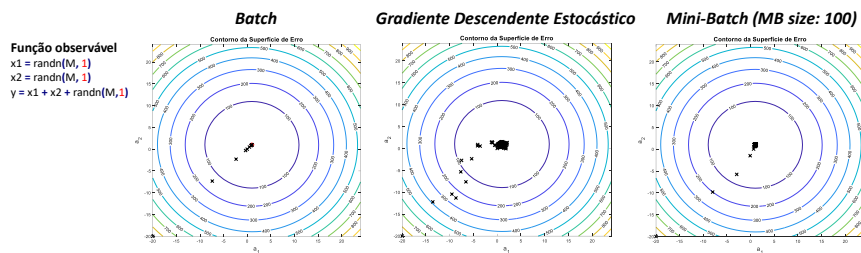
O mini-batch é bastante simples de entender quando você conhece o gradiente descendente em batelada e o gradiente descendente estocástico: a cada etapa, em vez de calcular os gradientes com base no conjunto de treinamento completo (como no GD em batelada) ou com base em apenas uma instância (como no GD estocástico), o mini-batch calcula os gradientes em subconjuntos aleatórios de instâncias chamados mini-lotes (do inglês mini-batch).

O progresso do algoritmo no espaço de parâmetros é menos irregular do que com o SGD, especialmente com mini lotes muito grandes. Como resultado, o mini-batch acabará chegando um pouco mais perto do mínimo do que o GDS. Mas, por outro lado, pode ser mais difícil escapar dos mínimos locais (no caso de problemas que sofrem com mínimos locais, diferentemente da Regressão Linear, que como vimos anteriormente apresenta apenas um mínimo, que é o global).

Exemplo: [mini_batch_gradient_descent_with_figures.ipynb](#)

Comparação das versões do GD

- Todos se aproximam do mínimo, mas o **batch** caminha diretamente em linha reta para lá.
- **GDE** e **mini-batch** ficam “dançando” ao redor do mínimo.
- O progresso do **mini-batch** é menos irregular do que o do **GDE**, mas depende do tamanho do mini-batch.
- Mas não se esqueçam, o **batch** leva muito tempo para executar cada época enquanto o **GDE** e **mini-batch** também alcançariam o mínimo caso uma boa estratégia para ajuste do passo de aprendizagem fosse usada.



O batch caminha diretamente em linha reta para o mínimo.

O progresso do mini-batch no espaço de parâmetros é menos irregular do que com o SGD, especialmente com mini-batches muito grandes.


No entanto, não se esqueça que o batch leva muito tempo para executar cada época, e o SGD e o mini-batch também alcançariam o mínimo se você usasse uma boa estratégia para modificação do passo de aprendizagem.

```
x1 = randn(M, 1);
x2 = randn(M, 1);
y = x1 + x2 + randn(M,1);
```

Tarefas

- **Quiz:** “*T319 - Quiz - Regressão: Parte II (1S2021)*” que se encontra no MS Teams.
- **Exercício Prático: Laboratório #3.**
 - Pode ser baixado do MS Teams ou do GitHub.
 - Pode ser respondido através do link acima (na nuvem) ou localmente.
 - [Instruções para resolução e entrega dos laboratórios.](#)

Obrigado!

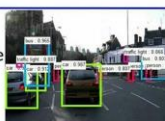



Deep Learning

linear regression

Online Courses

What they promise you will learn

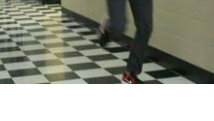




Linear Regression is NOT Machine Learning

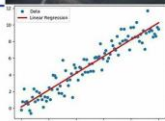
CHANGE MY MIND

WHEN YOU ADVERTISE IT'S ARTIFICIAL INTELLIGENCE. WHEN YOU HIRE IT'S MACHINE LEARNING.

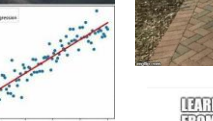


WHEN YOU IMPLEMENT IT'S LINEAR REGRESSION.

What you actually learn



WHY DON'T YOU JUST USE NORMAL LINEAR REGRESSION



LEARNING ML/DL FROM UNIVERSITY

ONLINE COURSES

FROM YOUTUBE

FROM ARTICLES

FROM MEMES

