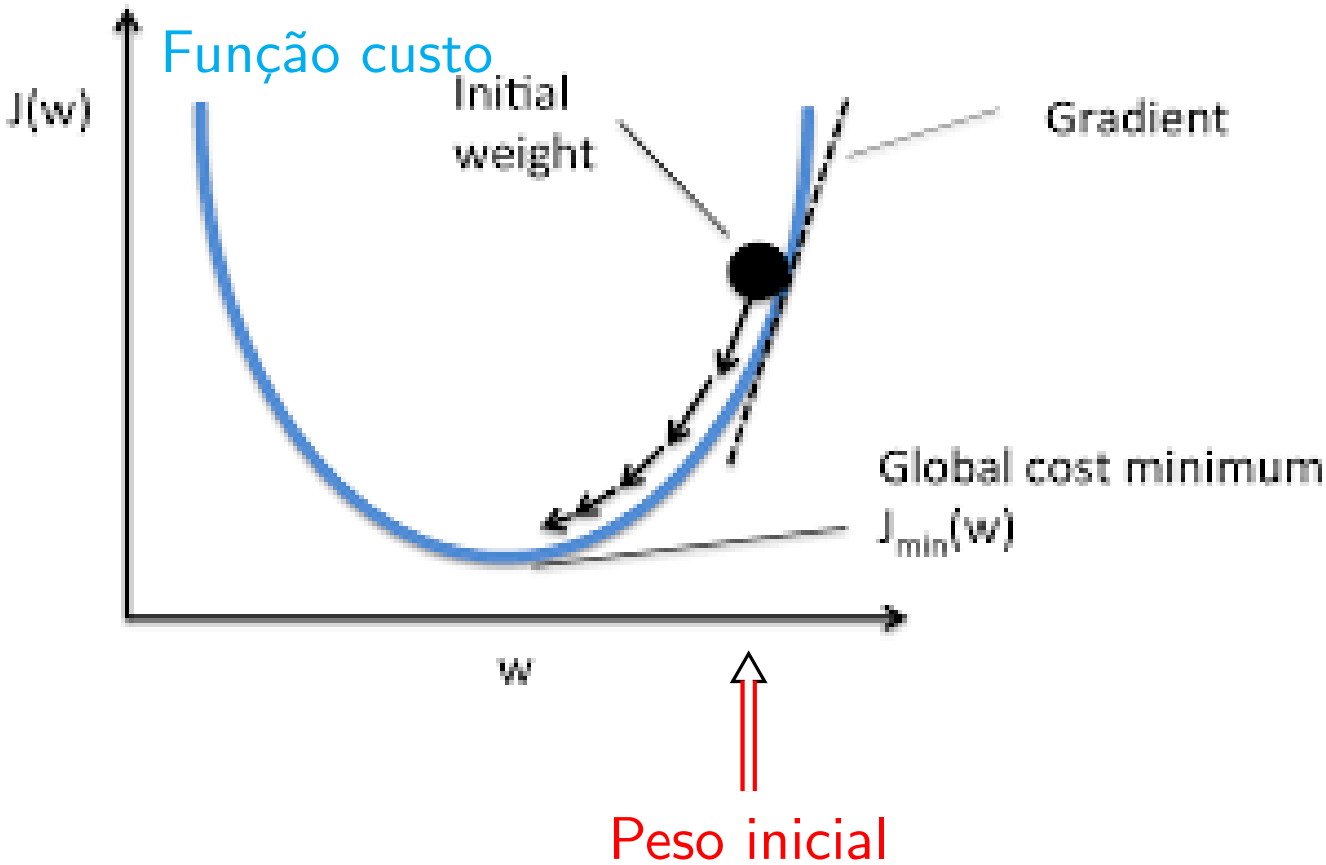
Gradient descent

Seja $J(\mathbf{w})$ a função custo a ser minimizada.

Algoritmo em linhas gerais

- chutar um valor para $\mathbf{w}$
- calcular o gradiente de J no ponto $\mathbf{w}$
  ("direção de maior inclinação")
- alterar $\mathbf{w}$ no sentido oposto ao do vetor gradiente

Ilustração do gradient descent



Exemplo: função custo usado em regressão linear

$$J(\mathbf{w}) = \frac{1}{N}\sum_{n=1}^{N}\left(y^{(n)} - \underbrace{h_{\mathbf{w}}(\mathbf{x}^{(n)})}_{\hat{y}^{(n)}=\mathbf{w}^T\mathbf{x}^{(n)}}\right)^2$$

- o que aqui está denotado J, no livro do Mostafa é E_{in}
- às vezes o $\frac{1}{N}$ não aparece; às vezes aparece um $\frac{1}{2}$ no lugar
- Lembre que pare este problema temos uma solução analítica (que usa a pseudo-inversa)

Vetor gradiente de J:

$$\nabla J(\mathbf{w}) = \left[\frac{\partial J}{\partial w_0}, \frac{\partial J}{\partial w_1}, \ldots, \frac{\partial J}{\partial w_d}\right]^T$$

$$\begin{aligned}
\frac{\partial J}{\partial w_j} &= \frac{\partial}{\partial w_j}\frac{1}{2}\sum_n (y^{(n)} - \hat{y}^{(n)})^2 \\
&= \frac{1}{2}\sum_n \frac{\partial}{\partial w_j}(y^{(n)} - \hat{y}^{(n)})^2 \\
&= \frac{1}{2}\sum_n 2(y^{(n)} - \hat{y}^{(n)})\frac{\partial}{\partial w_j}(y^{(n)} - \hat{y}^{(n)}) \\
&= \sum_n (y^{(n)} - \hat{y}^{(n)})\frac{\partial}{\partial w_j}\left(y_n - (w_0 + w_1 x_1^{(n)} + \ldots + w_j x_j^{(n)} + \ldots + w_d x_d^{(n)})\right) \\
&= -\sum_n (y^{(n)} - \hat{y}^{(n)})\, x_j^{(n)}
\end{aligned}$$

(o componente j do vetor gradiente depende do erro $(y^{(n)} - \hat{y}^{(n)})$ e os componentes j de todos os pontos $\mathbf{x}^{(n)}$)

Vetor gradiente de J: $\nabla J(\mathbf{w}) = \left[\frac{\partial J}{\partial w_0}, \frac{\partial J}{\partial w_1}, \ldots, \frac{\partial J}{\partial w_d}\right]$

$$\frac{\partial J}{\partial w_j} = -\sum_n (y^{(n)} - \hat{y}^{(n)})\, x_j^{(n)}$$

Peso inicial: $\mathbf{w}(0)$

Regra de atualização (iteração r):

$$\mathbf{w}(r+1) = \mathbf{w}(r) + \eta \Delta \mathbf{w}(r)$$

$$\Delta \mathbf{w}(r) = -\nabla J(\mathbf{w}), \qquad \Delta w_j(r) = \sum_n (y^{(n)} - \hat{y}^{(n)})\, x_j^{(n)}$$

η : taxa de aprendizado (em geral um valor pequeno)

Algorithm 1 GradientDescent

Input: D, η, *iter*

Output: $\mathbf{w}$

$\mathbf{w} \leftarrow$ small random value

repeat

$\Delta w_j \leftarrow 0, \quad j = 0, 1, 2, \ldots, d$

for all $(\mathbf{x}, y)$ in D **do**

compute $\hat{y} = \mathbf{w}^T \mathbf{x}$

$\Delta w_j \leftarrow \Delta w_j + (y - \hat{y})\, x_j, \quad j = 0, 1, 2, \ldots, d$

end for

$w_j \leftarrow w_j + \eta \Delta w_j, \quad j = 0, 1, 2, \ldots, d$

until number of iterations = *iter*

Algorithm 2 Stochastic GradientDescent

Input: D, η, *iter*

Output: $\mathbf{w}$

$\mathbf{w} \leftarrow$ small random value

repeat

for all $(\mathbf{x}, y)$ in D **do**

compute $\hat{y} = \mathbf{w}^T \mathbf{x}$

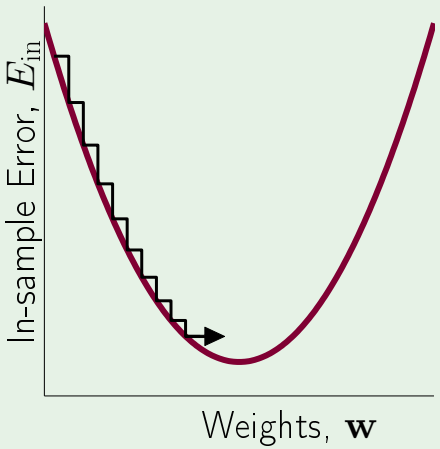
$w_j \leftarrow w_j + \eta(y - \hat{y})\, x_j, \quad j = 0, 1, 2, \ldots, d$

end for

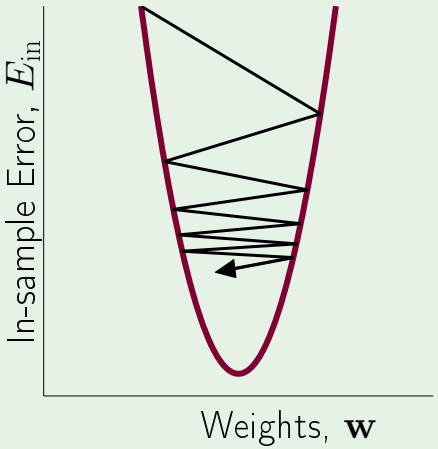
until number of iterations $=$ *iter*

Fixed-size step?

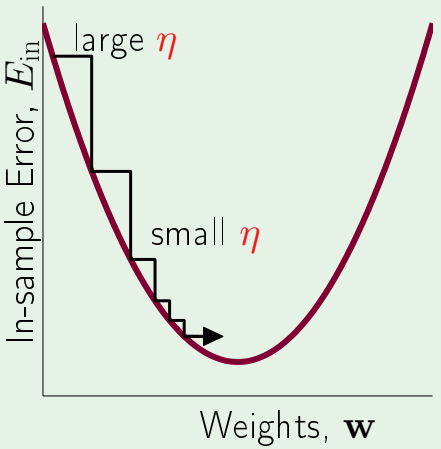
How η affects the algorithm:



η too small



η too large



variable η – just right

η should increase with the slope

Observation: the example shown previously is the application of the gradient descent to the MSE (mean squared error) cost function. The algorithm can be applied to minimize other cost functions (and, obviously, the gradient vector shoul be computed accordingly).

Mais um pouco sobre

Gradient Descent $\times$ **Stochastic gradient descent**

Stochastic gradient descent

GD minimizes:

$$E_{\text{in}}(\mathbf{w}) = \frac{1}{N}\sum_{n=1}^{N} \underbrace{\mathbf{e}\left(h(\mathbf{x}_n), y_n\right)}_{\ln\left(1+e^{-y_n \mathbf{w}^{\mathsf{T}}\mathbf{x}_n}\right)} \longleftarrow \text{in logistic regression}$$

by iterative steps along $-\nabla E_{\text{in}}$:

$$\Delta\mathbf{w} = -\eta\, \nabla E_{\text{in}}(\mathbf{w})$$

∇E_{in} is based on all examples $(\mathbf{x}_n, y_n)$

"**batch**" GD

The stochastic aspect

Pick one $(\mathbf{x}_n, y_n)$ at a time. Apply GD to $\mathbf{e}\left(h(\mathbf{x}_n), y_n\right)$

"Average" direction:

$$\mathbb{E}_n\left[-\nabla \mathbf{e}\left(h(\mathbf{x}_n), y_n\right)\right] = \frac{1}{N}\sum_{n=1}^{N} -\nabla \mathbf{e}\left(h(\mathbf{x}_n), y_n\right)$$

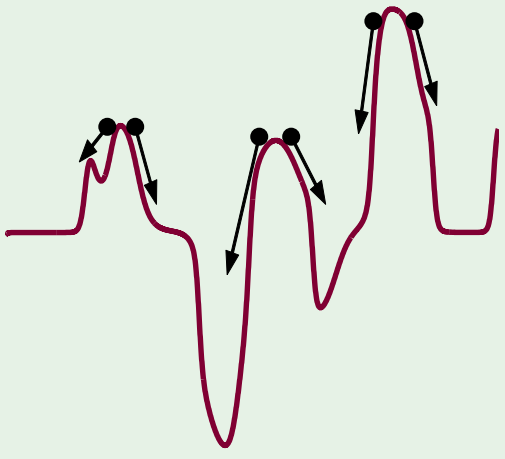
$$= -\nabla E_{\text{in}}$$

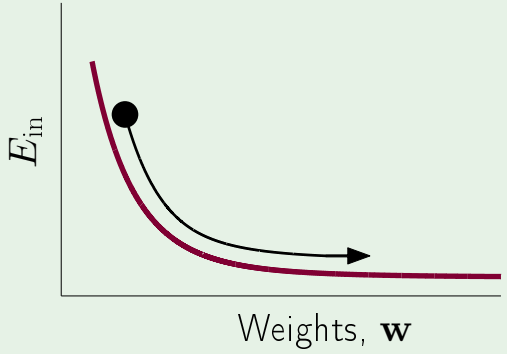
randomized version of GD

stochastic gradient descent (SGD)

Benefits of SGD

1. cheaper computation
2. randomization
3. simple



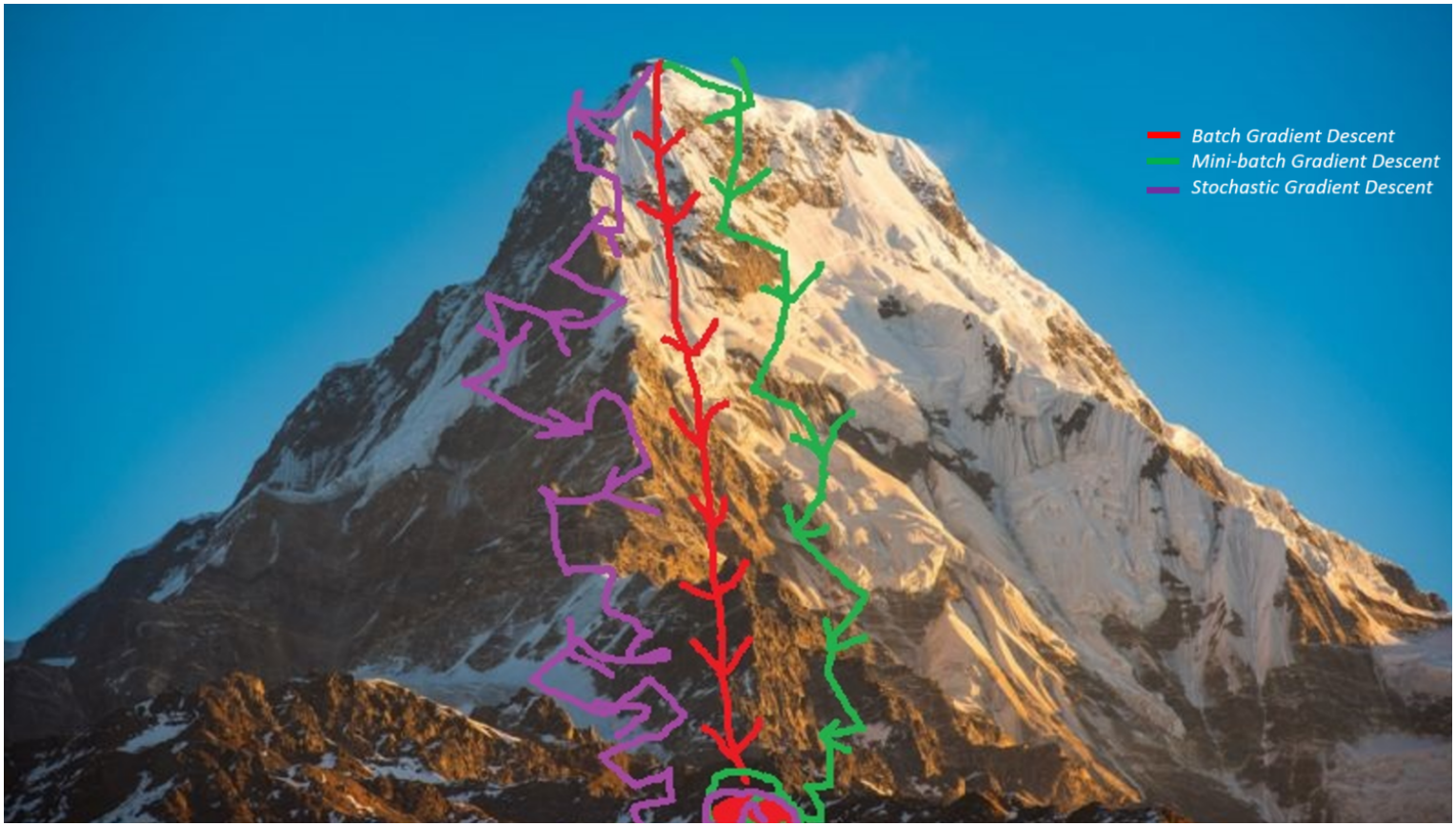


randomization helps

Rule of thumb:

$\eta = 0.1$ works

Gradient descent



https://imaddabbura.github.io/post/gradient_descent_algorithms/