Deep Learning

Sem 02 - Aula 03

Renato Assunção - DCC - UFMG

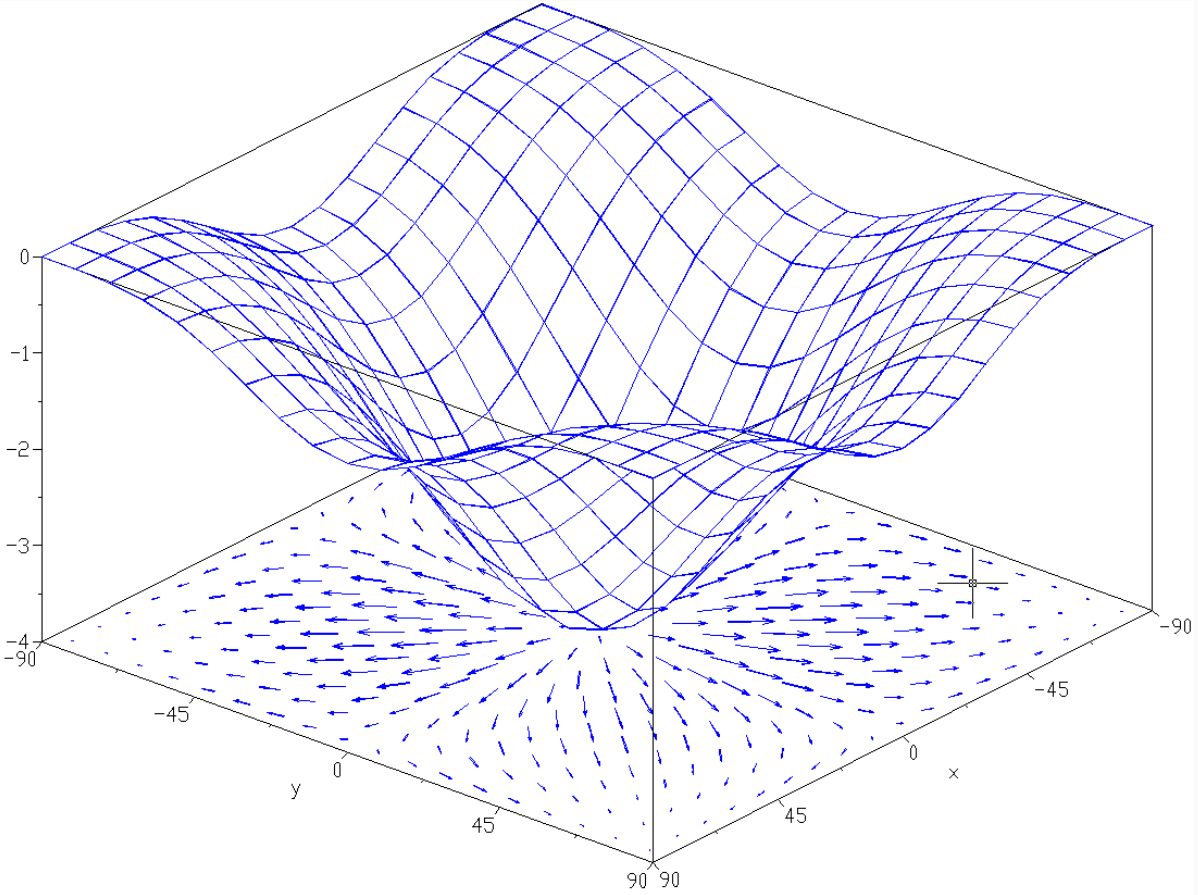
Vetor Gradiente de J(x,y)

Gradiente é um vetor associado com cada ponto (x,y) do plano.

Gradiente muda com o ponto (x,y)

Ele aponta na direção de crescimento máximo da função J(x,y)

Ele mostra para onde mexer LIGEIRAMENTE no ponto (x,y) para aumentar J(x,y) de forma maximal



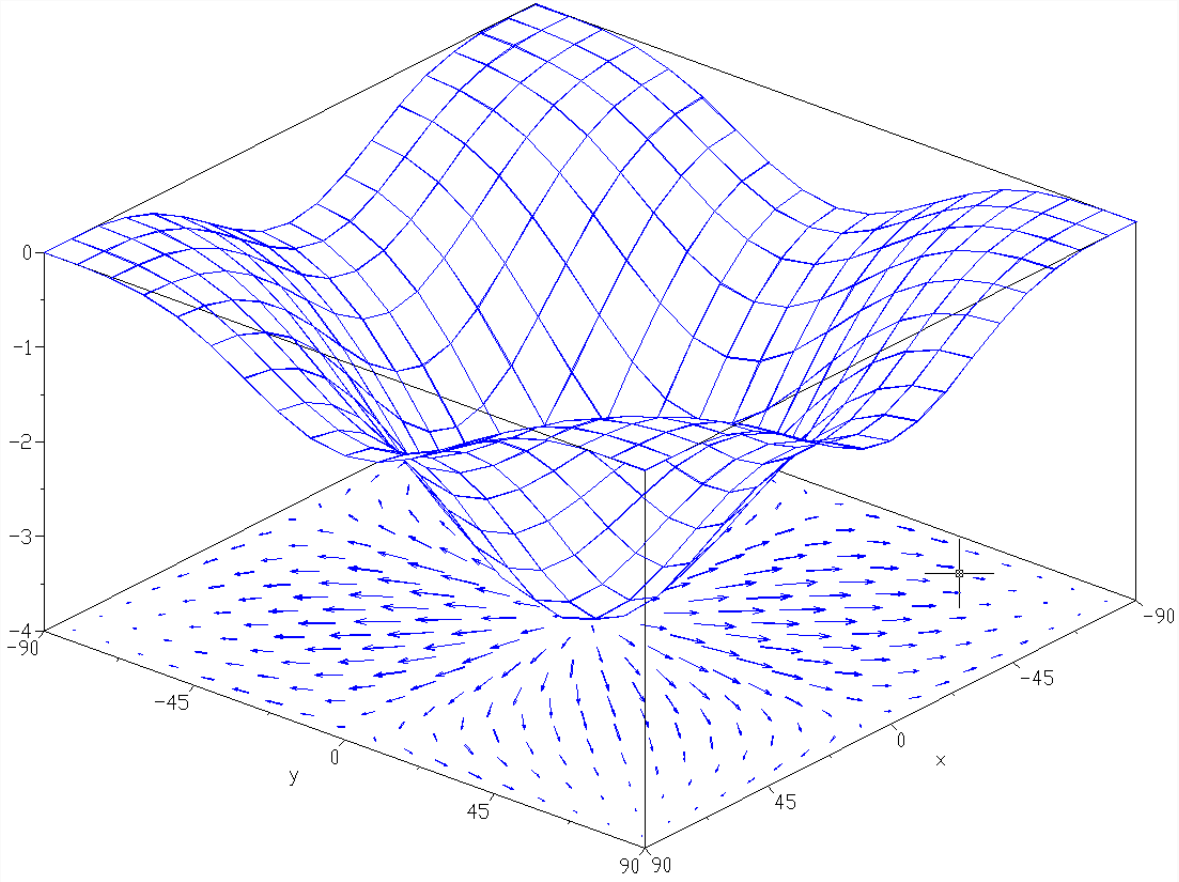
Vetor Gradiente

Para procurar o mínimo, andamos então na direção OPOSTA ao gradiente da função de custo J

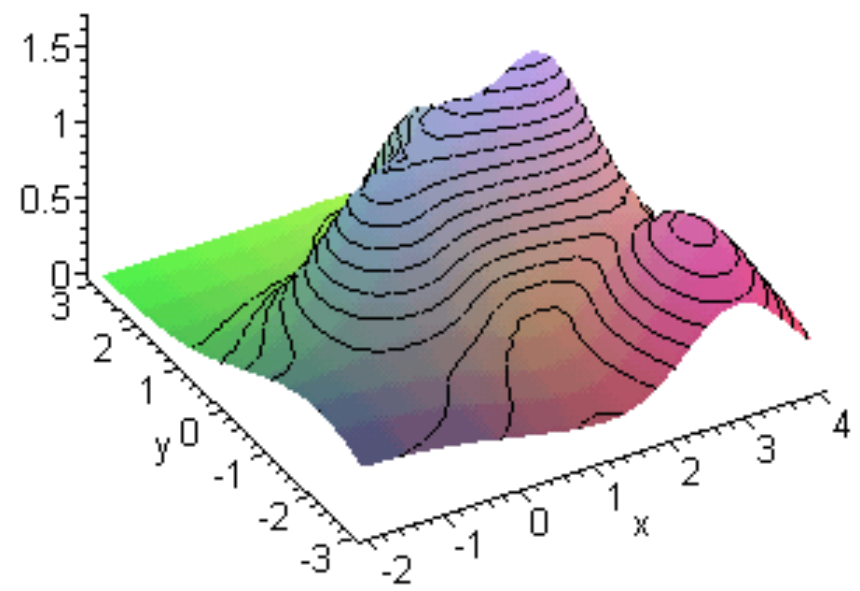
Isto é, andamos na direção:

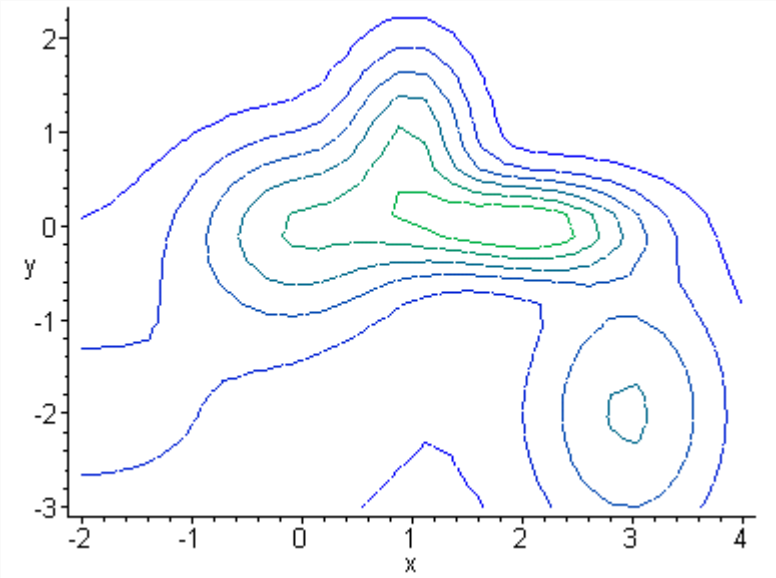
contrária àquela

apontada pelo Gradiente

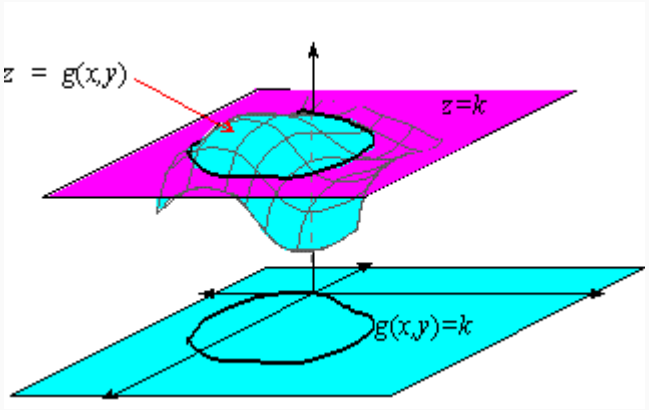


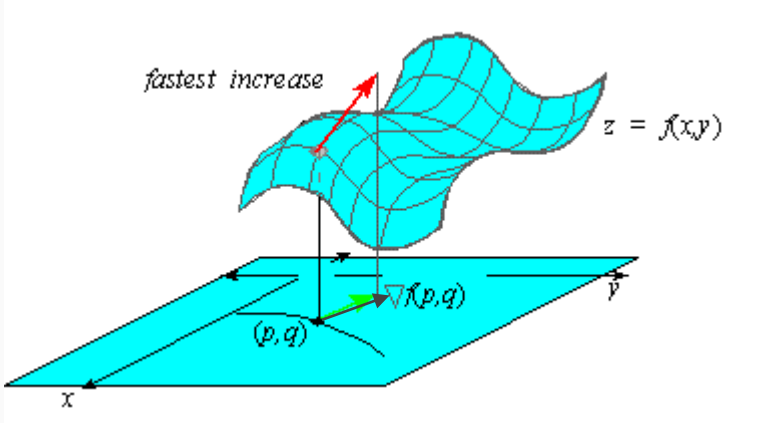
J(x,y) e as curvas de nível





Curvas de nível e o gradiente

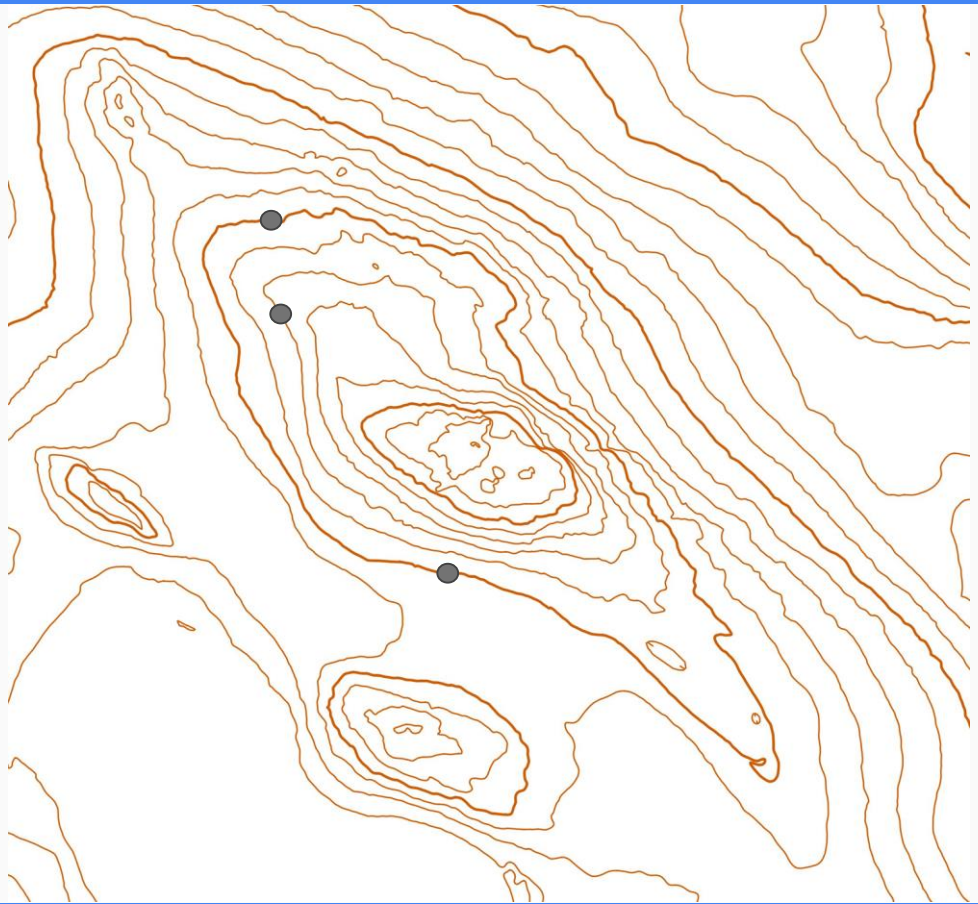




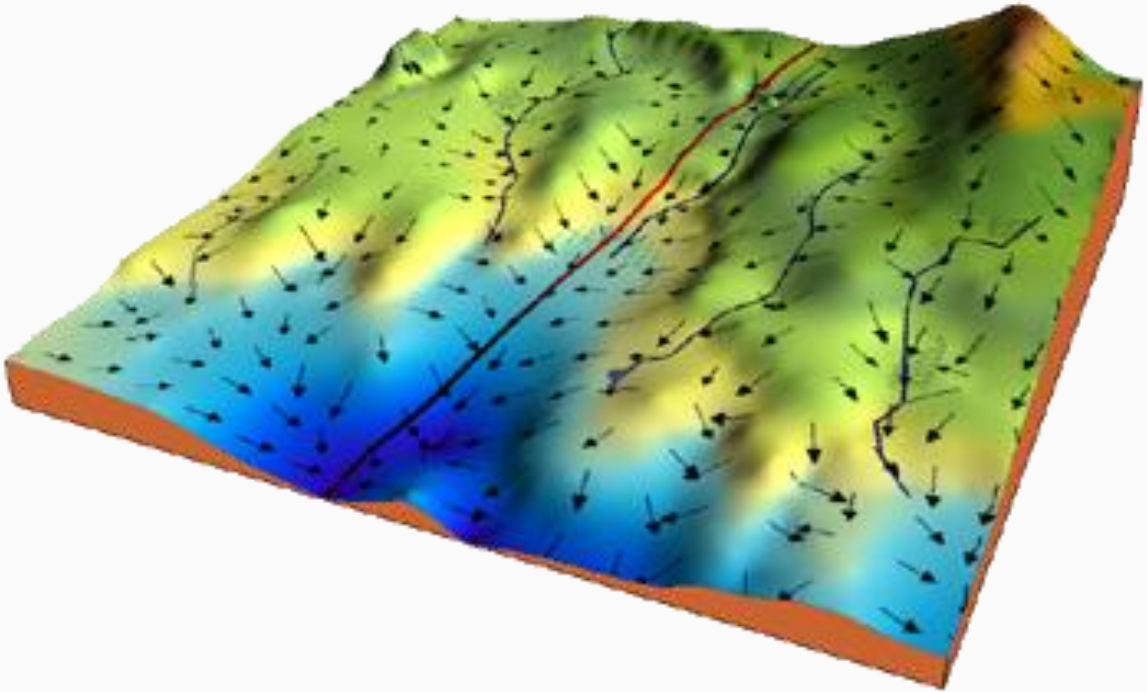
Gradiente:

- é um vetor associado com cada ponto (x,y) do plano.
- Muda com o ponto (x,y)
- Aponta na direção em que devemos mexer no ponto (x,y) para aumentar J(x,y) de forma maximal
- Gradiente é **ortogonal** à curva de nível que passa por (x,y)

Encontre o vetor gradiente nos pontos abaixo



Gradiente descendente



As setas representam a direção de descida mais íngreme (gradiente negativo) num certo ponto.

É a direção que diminui ao máximo a função de custo se dermos um pequeno passo saindo da posição (x,y).

Esqueça o hessiano, use apenas o gradiente

- Ao invés de usar

$$\mathbf{w}^{k+1} = \begin{bmatrix} w_0^{k+1} \\ w_1^{k+1} \\ \vdots \\ w_n^{k+1} \end{bmatrix} = \begin{bmatrix} w_0^{k} \\ w_1^{k} \\ \vdots \\ w_n^{k} \end{bmatrix} - \Big[\underbrace{J^2(\mathbf{w}^k)}_{\text{matriz der. parciais 2a ordem de J}} \Big]^{-1} \underbrace{\nabla J(\mathbf{w}^k)}_{\text{vetor gradiente de J}}$$

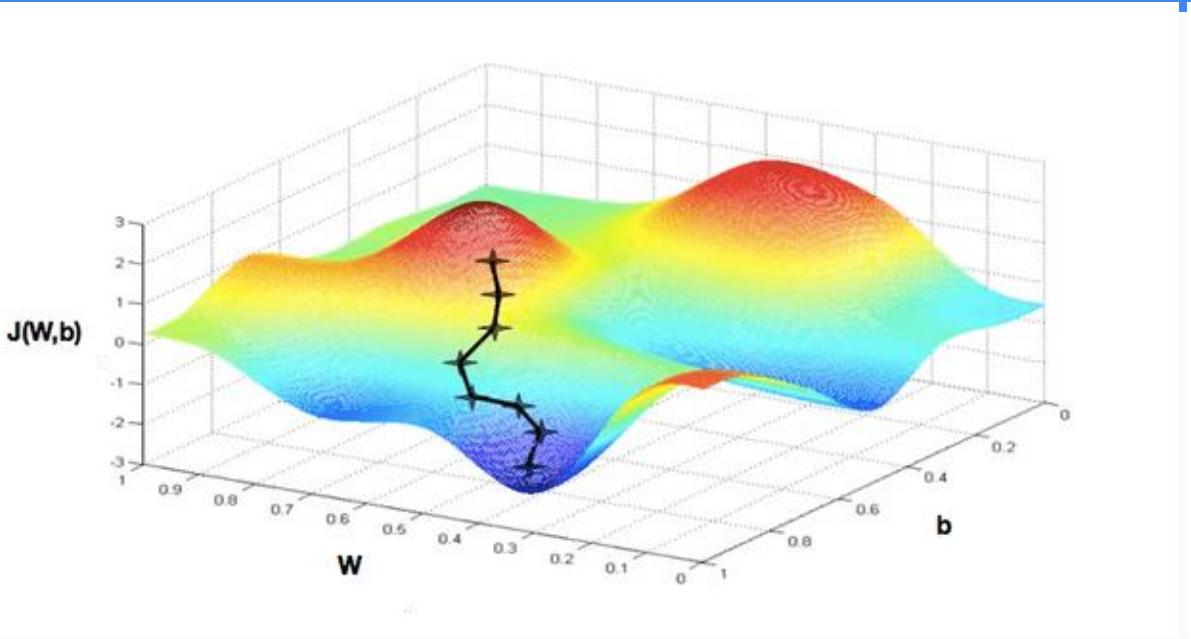
- usamos

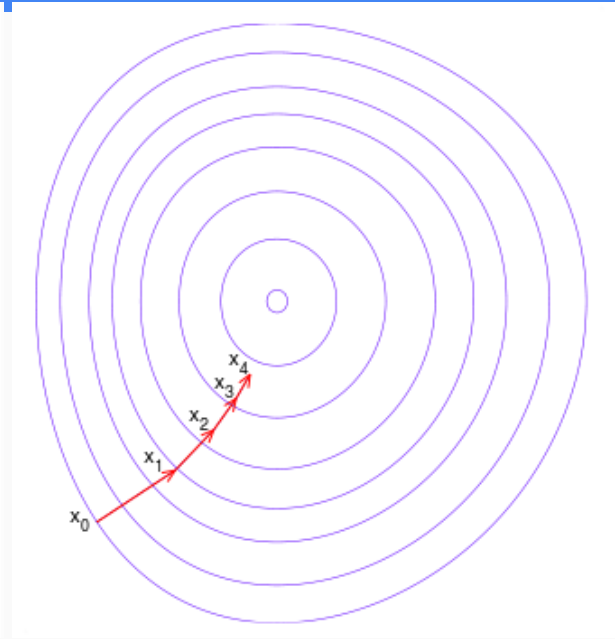
$$\mathbf{w}^{k+1} = \begin{bmatrix} w_0^{k+1} \\ w_1^{k+1} \\ \vdots \\ w_n^{k+1} \end{bmatrix} = \begin{bmatrix} w_0^{k} \\ w_1^{k} \\ \vdots \\ w_n^{k} \end{bmatrix} - \alpha \underbrace{\nabla J(\mathbf{w}^k)}_{\text{vetor gradiente de J}}$$

α = learning rate
é um escalar positivo e pequeno.

Por exemplo, é comum usar $\alpha = 0.01$

Caminho ideal via método do gradiente descendente

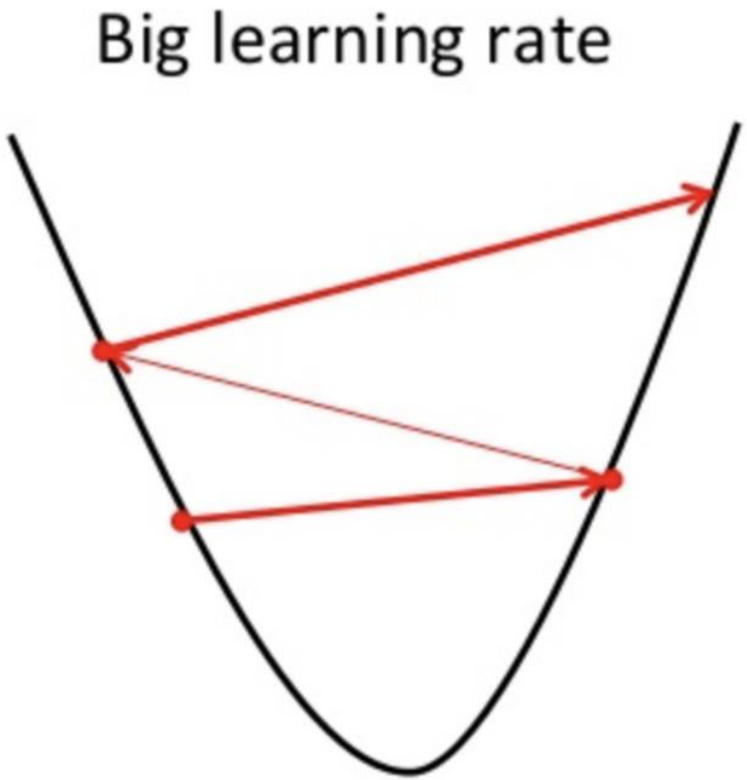


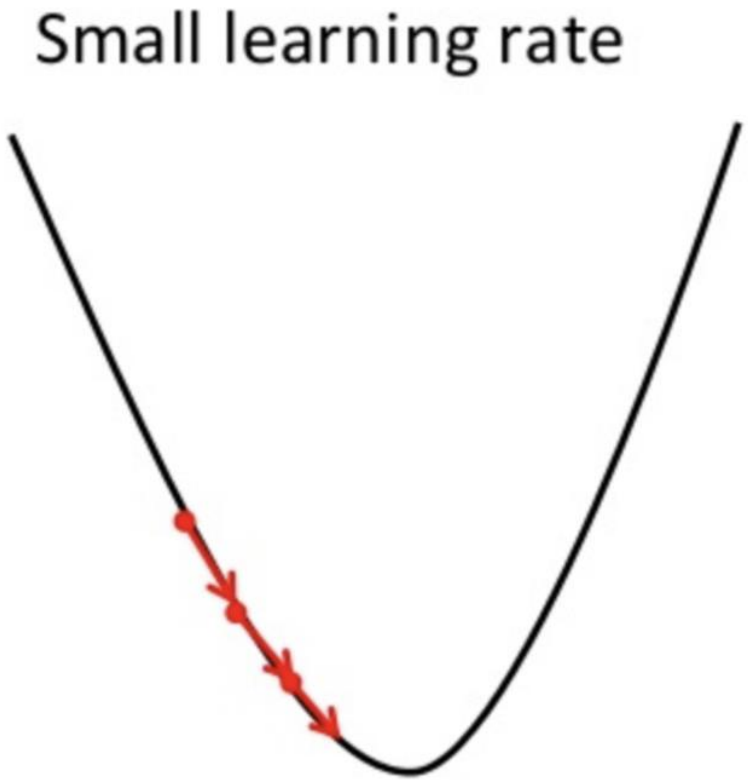


- Quando a função f(x,y) é bem aproximada por um parabolóide circular em torno do ponto de mínimo:

- neste caso, o gradiente descendente vai funcionar maravilhosamente se:
  alpha é pequeno o suficiente para evitar overshooting (ultrapassar o ponto de mínimo)
  alpha não é tão pequeno a ponto das atualizações quase não se mexerem.

A escolha de alpha = learning rate

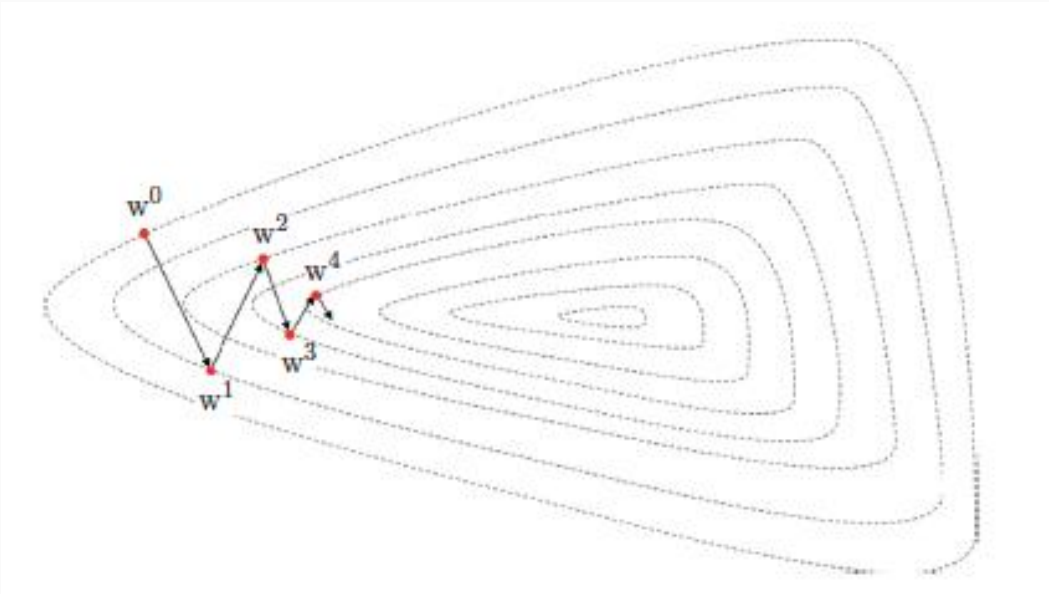




Um problema prático

Em dimensões mais altas (muitos parâmetros), MESMO COM um bom learning rate:

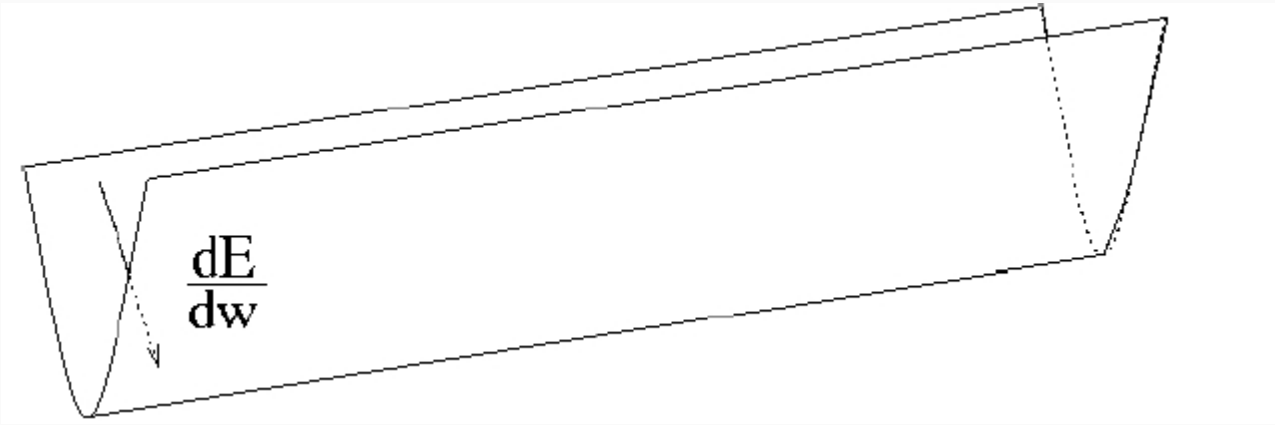
os passos sucessivos do gradiente descendente tendem a ziguezaguear em direção ao mínimo



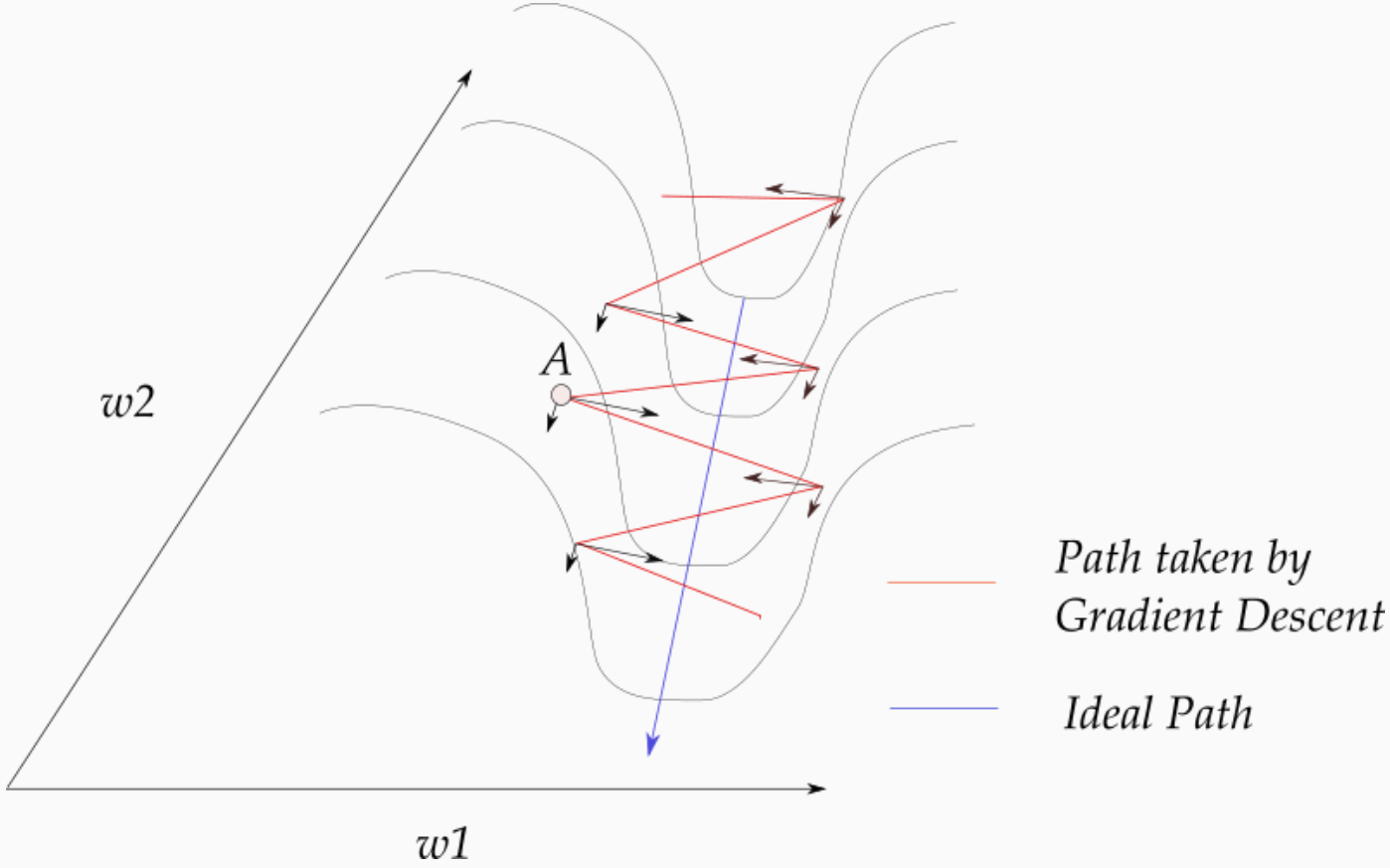
As direções sucessivas do gradiente descendente são perpendiculares aos contornos das curvas de nível.

Quando isto pode ocorrer?

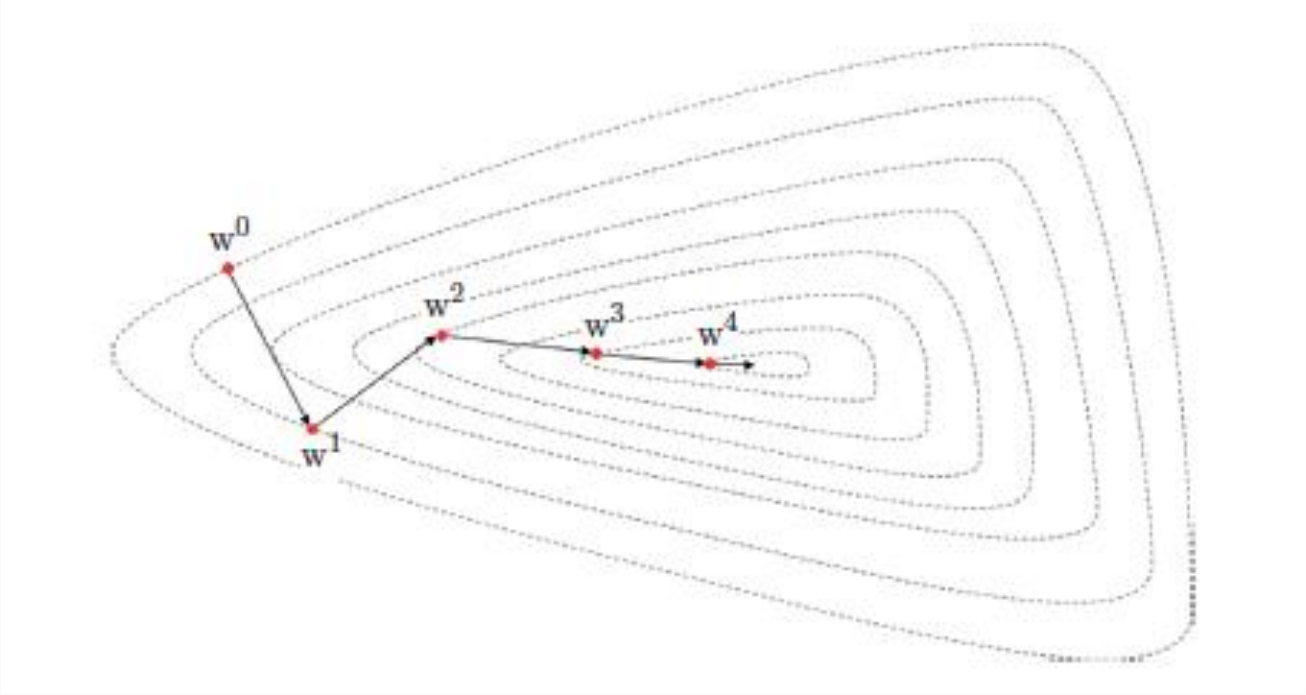
- Função de custo $J(\mathbf{w})$
- Imagine que a superfície da função de custo $J(\mathbf{w})$ seja na forma de um vale longo e estreito que desce lentamente em direção ao ponto de mínimo.
- O gradiente descendente desce rapidamente pelas paredes do vale, mas vai muito lentamente para o longo fundo do vale.



Gradiente em regiões de vales descendentes



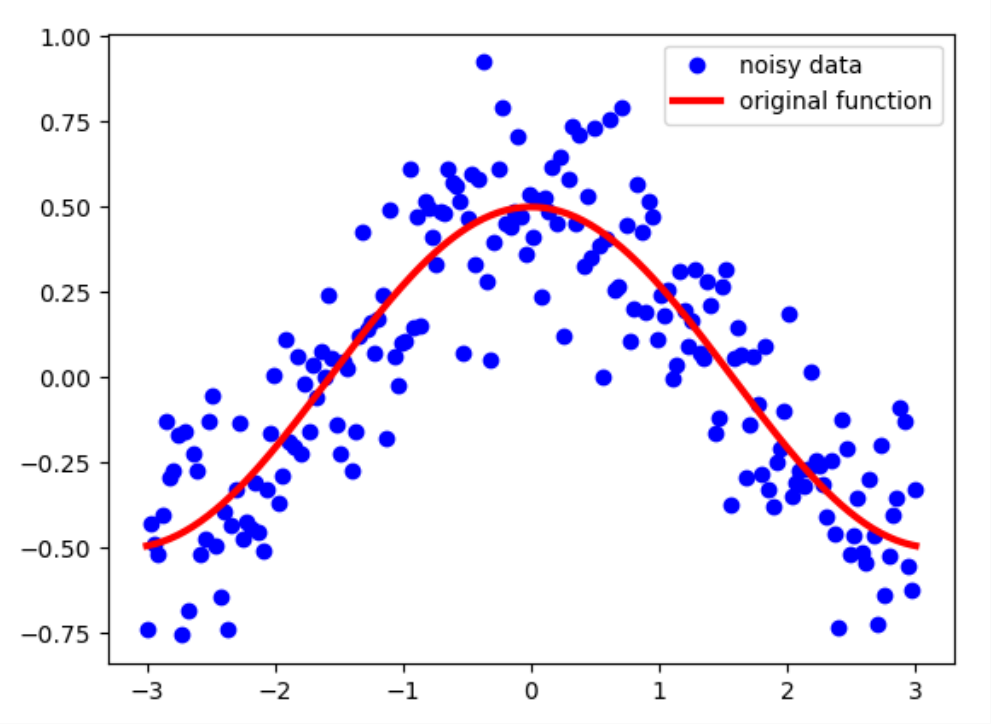
Momentum: uma solução parcial



Gradiente descendente corrigido pelo método de momento: zig-zag atenuado.

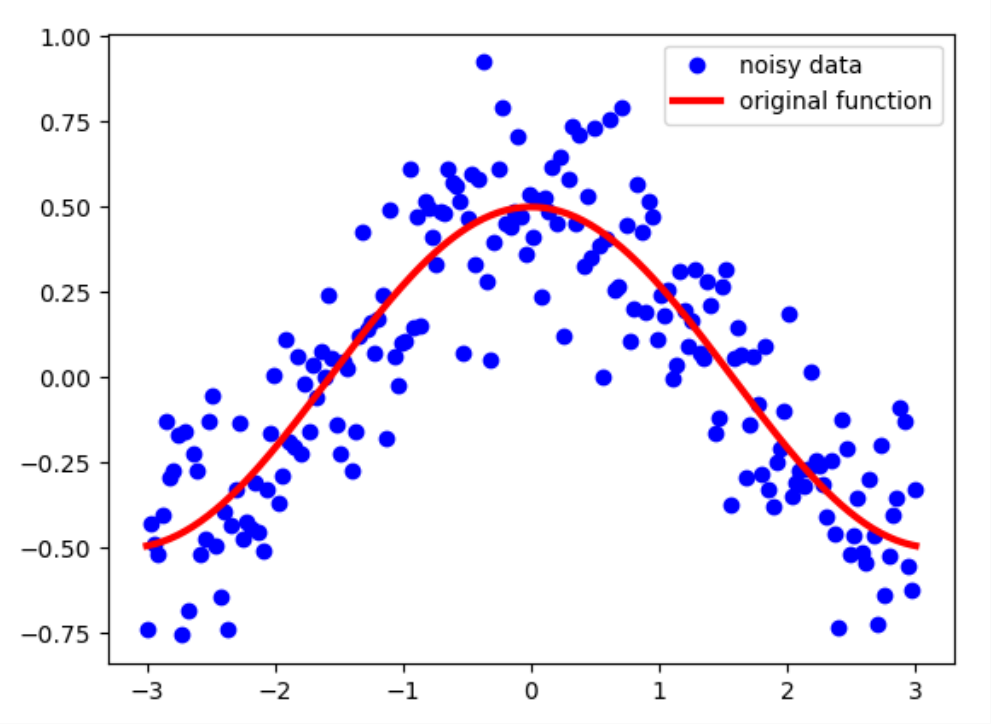
Como isto é feito? Com exponential smoothing ou exponential weighted average

Exponential smoothing para estimar uma curva

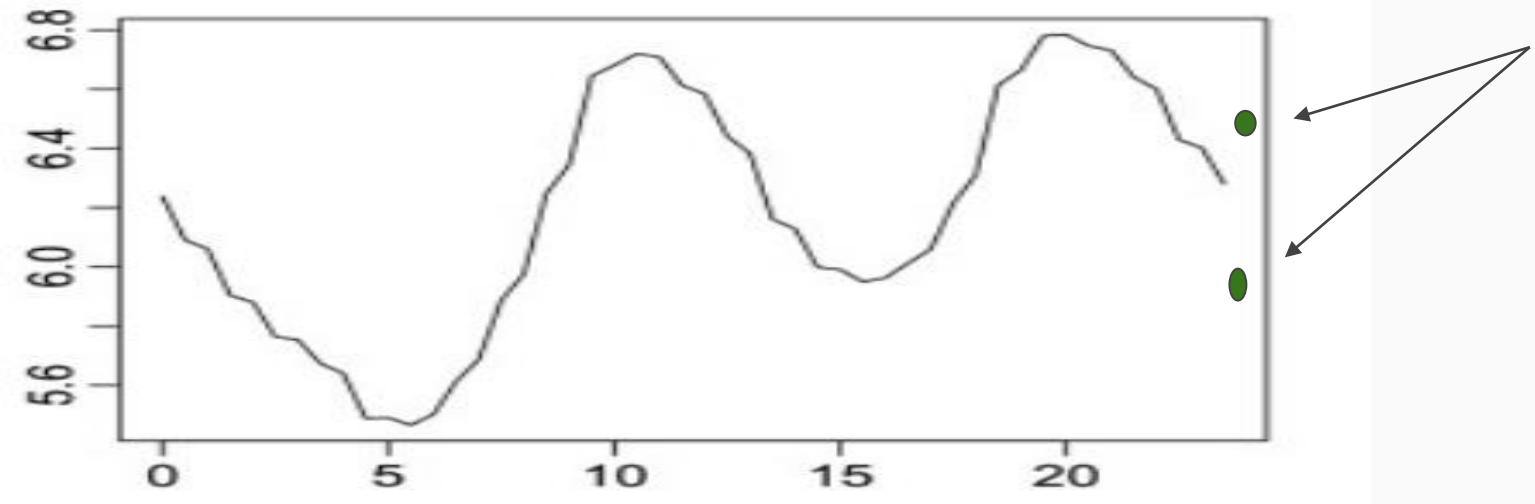


- Função seno em vermelho
- Dados de série temporal: pontos azuis
- dados = curva + ruído → objetivo é recuperar a curva vermelha a partir dos dados com ruído

Exponential smoothing para estimar uma curva



- Imagine que num tempo t-1 qualquer você tenha uma boa estimativa da curva vermelha: V_{t-1}
- Como atualizar V_{t-1} com o novo dado S_t ?

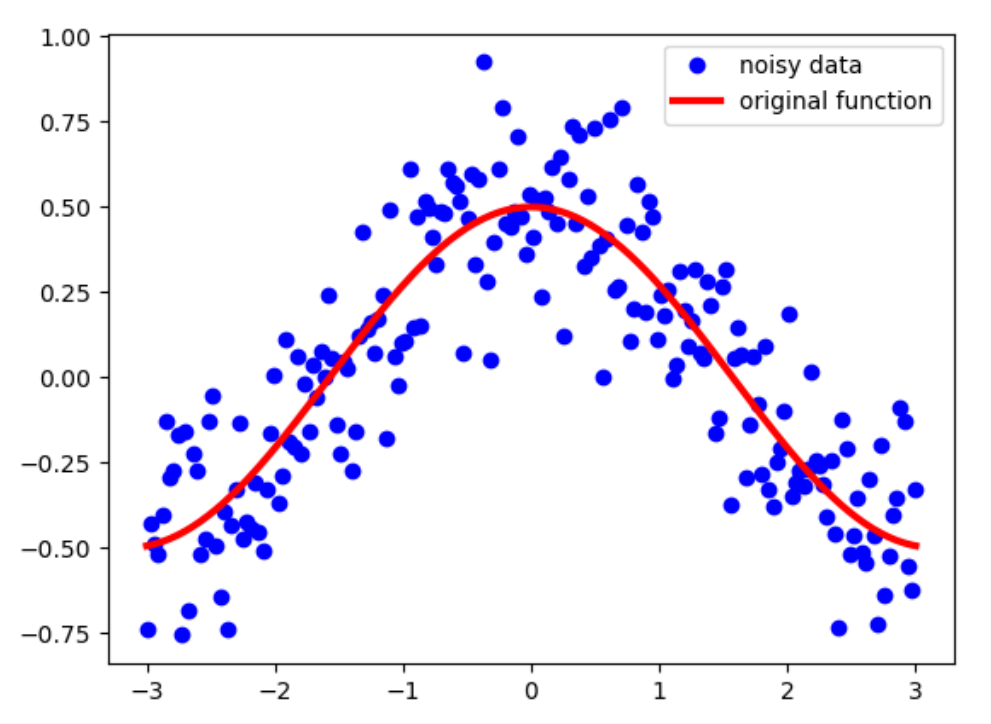


Novo dado S_t informa a tendência mais recente da série.

Se S_t subir muito, puxe a série para cima

Se S_t descer muito, puxe para baixo. Mas de quanto devemos puxar??

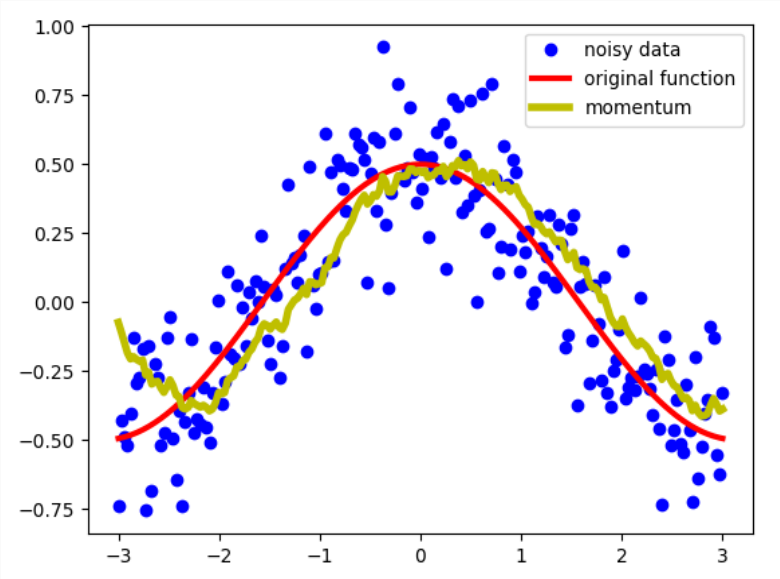
Exponential smoothing para estimar uma curva



- Em t-1 você tenha uma boa estimativa da curva vermelha: V_{t-1}
- Chega o novo dado S_t
- Mude um pouco V_{t-1} na direção de S_t : $V_{t-1} = 0.9V_t + 0.1S_t$

Média ponderada entre a (boa) estimativa até agora e o novo dado

Exponential smoothing para estimar uma curva



$$V_1 = S_1$$
$$V_2 = 0.9V_1 + 0.1S_2$$
$$V_3 = 0.9V_2 + 0.1S_3$$
$$\vdots$$
$$V_t = 0.9V_{t-1} + 0.1S_t$$

- Existe uma inércia em mudar V_t
- Ele resiste com 90% do seu valor anterior

Exponential smoothing para estimar uma curva

$$
\begin{aligned}
V_1 &= S_1 \\
V_2 &= 0.9V_1 + 0.1S_2 \\
&= 0.9S_1 + 0.1S_2 \\
V_3 &= 0.9V_2 + 0.1S_3 \\
&= (0.9)^2 S_1 + (0.9)(0.1)S_2 + 0.1S_3 \\
&\vdots \\
V_t &= 0.9V_{t-1} + 0.1S_t \\
&= (0.9)^{t-1} S_1 + (0.9)^{t-2}(0.1)S_2 + (0.9)^{t-3}(0.1)S_3 + \ldots + (0.9)(0.1)S_{t-1} + (0.1)S_t
\end{aligned}
$$

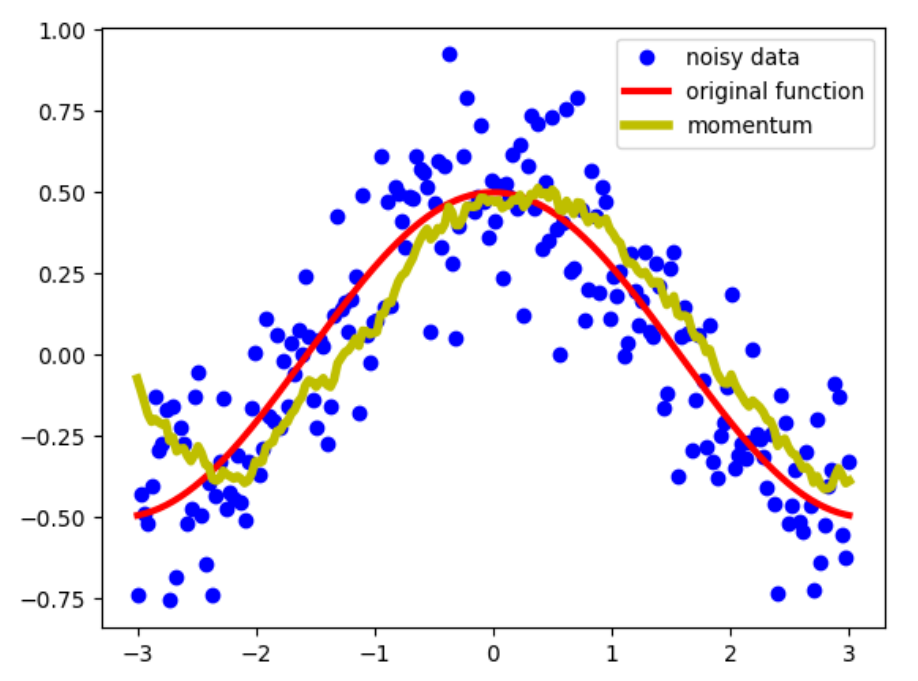
$$
V_t = (0.1)\left[S_t + (0.9)S_{t-1} + (0.9)^2 S_{t-2} + (0.9)^3 S_{t-3} + (0.9)^4 S_{t-4} + \ldots\right]
$$

Pesos somam aproximadamente 1

Pesos somam = 1 - 0.9^{t}

0.9^{20} = 0.12 0.9^{50} = 0.005 0.9^{100} = 0.000027

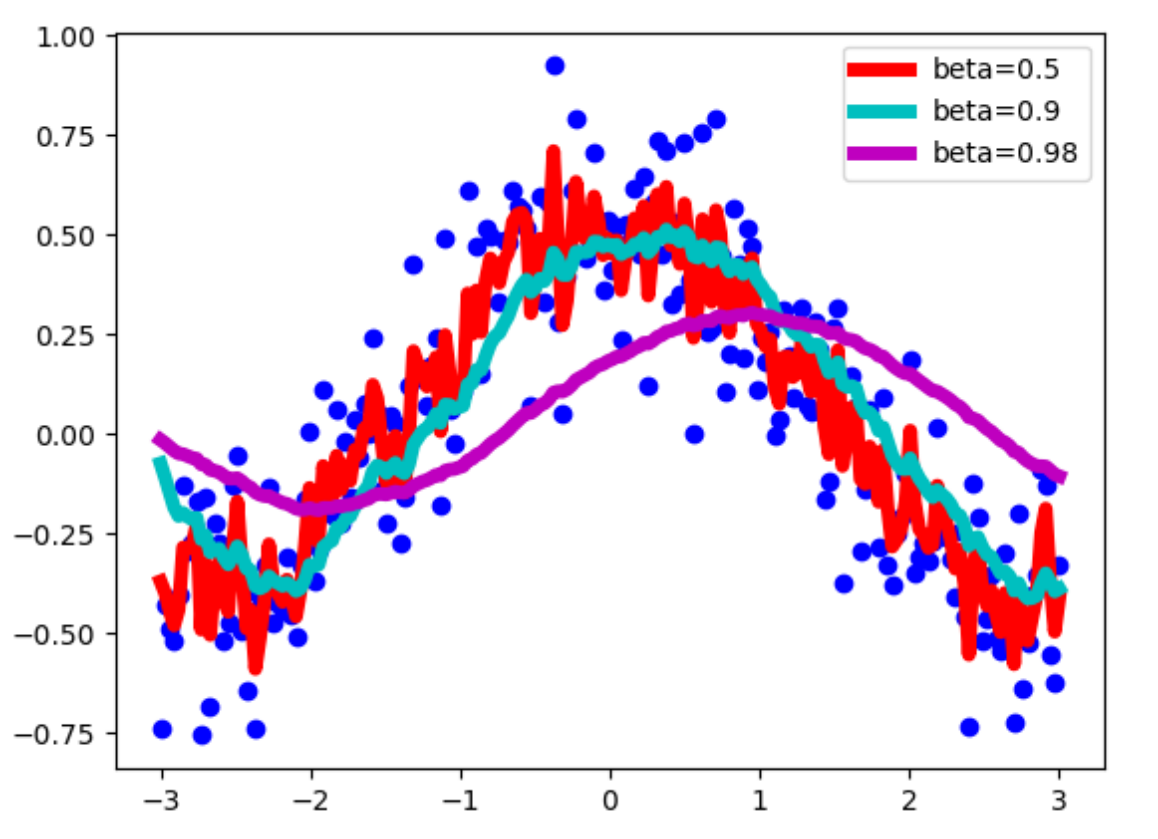
Exponential smoothing para estimar uma curva



Usamos $\beta = 0.9$
Podemos deixar um beta genérico

$$V_1 = S_1$$
$$V_t = \beta V_{t-1} + (1 - \beta) S_t$$

Impacto da escolha de beta



Momentum com backpropagation

- Gradiente descendente:

$$\mathbf{w}^{(t+1)} = \begin{bmatrix} w_0^{(t+1)} \\ w_1^{(t+1)} \\ \vdots \\ w_n^{(t+1)} \end{bmatrix} = \begin{bmatrix} w_0^{(t)} \\ w_1^{(t)} \\ \vdots \\ w_n^{(t)} \end{bmatrix} - \alpha \underbrace{\nabla \mathcal{L}(\mathbf{w}^{(t)})}_{\text{vetor gradiente}}$$

- Aliviamos o problema do gradiente oscilante atualizando os parâmetros com um gradiente suavizado como se fosse uma série temporal.
- Suponha que, depois de (t) iterações, tenhamos um ponto $\mathbf{w}^{(t)}$
- Avaliamos o gradiente neste ponto obtendo $\nabla \mathcal{L}(\mathbf{w}^{(t)})$

- Baseado nos w's anteriores (até t-1), suponha que tenhamos uma aproximação razoável para o série temporal do gradiente dada por certo $\mathbf{V}^{(t-1)}$
- Esta boa aproximação é usada numa média ponderada com o gradiente mais recente

Momentum com backpropagation

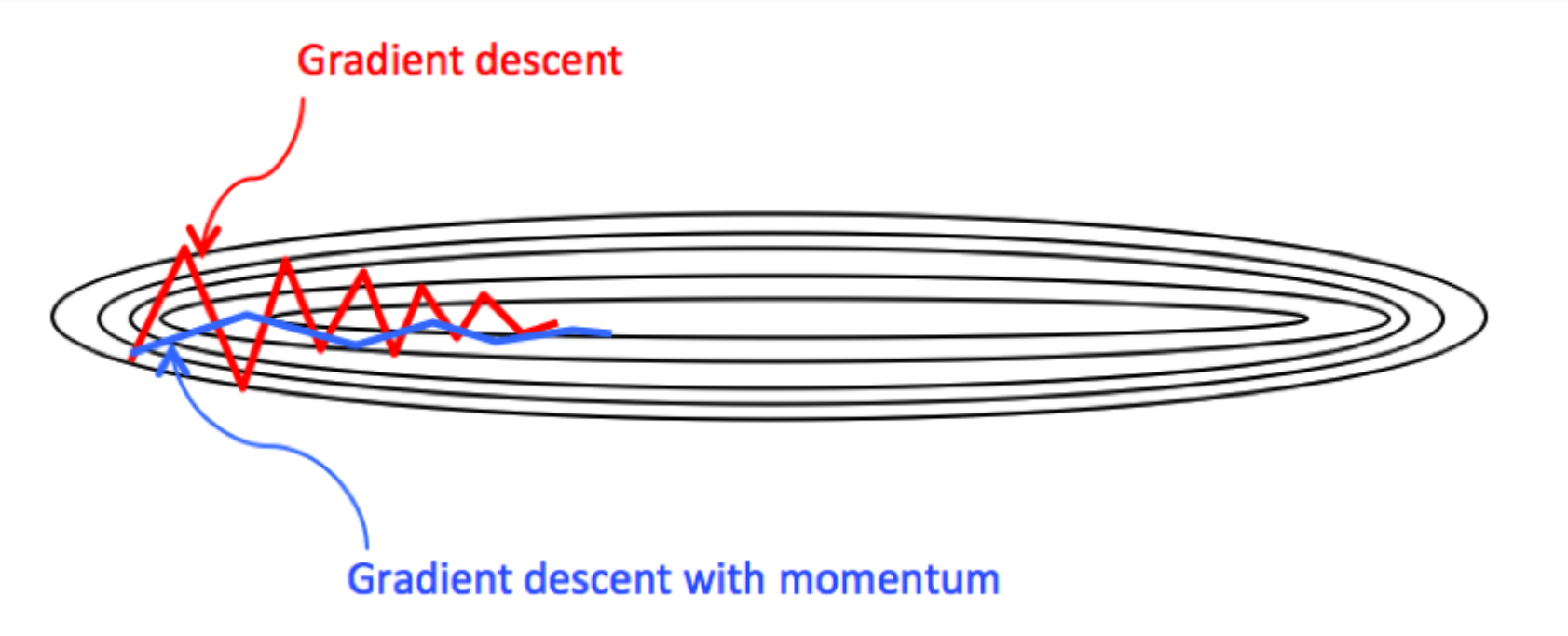
- Gradiente descendente:

$$\mathbf{w}^{(t+1)} = \begin{bmatrix} w_0^{(t+1)} \\ w_1^{(t+1)} \\ \vdots \\ w_n^{(t+1)} \end{bmatrix} = \begin{bmatrix} w_0^{(t)} \\ w_1^{(t)} \\ \vdots \\ w_n^{(t)} \end{bmatrix} - \alpha \underbrace{\nabla \mathcal{L}(\mathbf{w}^{(t)})}_{\text{vetor gradiente}}$$

- Momentum: usando $\nabla \mathcal{L}(\mathbf{w}^{(0)}) = \mathbf{V}^{(0)}$

$$\mathbf{w}^{(t+1)} = \begin{bmatrix} w_0^{(t+1)} \\ w_1^{(t+1)} \\ \vdots \\ w_n^{(t+1)} \end{bmatrix} = \begin{bmatrix} w_0^{(t)} \\ w_1^{(t)} \\ \vdots \\ w_n^{(t)} \end{bmatrix} - \alpha \underbrace{\left[(1-\beta) \nabla \mathcal{L}(\mathbf{w}^{(t)}) + \beta \mathbf{V}^{(t-1)} \right]}_{\mathbf{V}^{(t)}}$$

Resultado



Nesterov

- Uma modificação adicional que melhora o desempenho: Nesterov
- Momentum:

$$\mathbf{w}^{(t+1)} = \begin{bmatrix} w_0^{(t+1)} \\ w_1^{(t+1)} \\ \vdots \\ w_n^{(t+1)} \end{bmatrix} = \begin{bmatrix} w_0^{(t)} \\ w_1^{(t)} \\ \vdots \\ w_n^{(t)} \end{bmatrix} - \alpha \underbrace{\left[(1-\beta) \nabla \mathcal{L}(\mathbf{w}^{(t)}) + \beta \mathbf{V}^{(t-1)} \right]}_{\mathbf{V}^{(t)}}$$

- Não usa o gradiente em $\mathbf{w}^{(t)}$
- Vamos tentar obter uma posição projetada no próximo passo, ANTES de dar este passo.
- Temos uma “boa estimativa” da série do gradiente neste momento: $\mathbf{V}^{(t-1)}$
- Projetamos então para o próximo passo: $\mathbf{w}^{(t)} - \alpha \mathbf{V}^{(t-1)}$
- É neste ponto que calculamos o gradiente

Nesterov

- Momentum:

$$\mathbf{w}^{(t+1)} = \begin{bmatrix} w_0^{(t+1)} \\ w_1^{(t+1)} \\ \vdots \\ w_n^{(t+1)} \end{bmatrix} = \begin{bmatrix} w_0^{(t)} \\ w_1^{(t)} \\ \vdots \\ w_n^{(t)} \end{bmatrix} - \alpha \underbrace{\left[(1-\beta) \nabla \mathcal{L}(\mathbf{w}^{(t)}) + \beta \mathbf{V}^{(t-1)} \right]}_{\mathbf{V}^{(t)}}$$

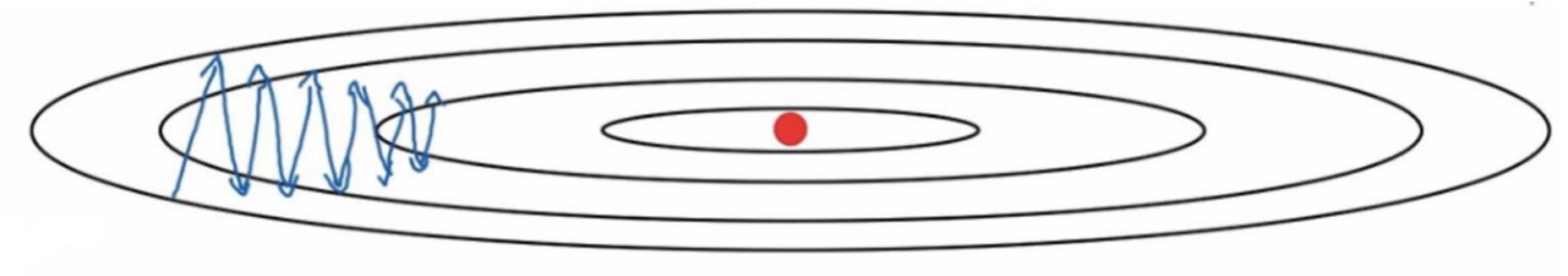
- Nesterov

$$\mathbf{w}^{(t+1)} = \begin{bmatrix} w_0^{(t+1)} \\ w_1^{(t+1)} \\ \vdots \\ w_n^{(t+1)} \end{bmatrix} = \begin{bmatrix} w_0^{(t)} \\ w_1^{(t)} \\ \vdots \\ w_n^{(t)} \end{bmatrix} - \alpha \underbrace{\left[(1-\beta) \nabla \mathcal{L}(\mathbf{w}^{(t)} - \alpha \mathbf{V}^{(t-1)}) + \beta \mathbf{V}^{(t-1)} \right]}_{\mathbf{V}^{(t)}}$$

O mais popular com backprop

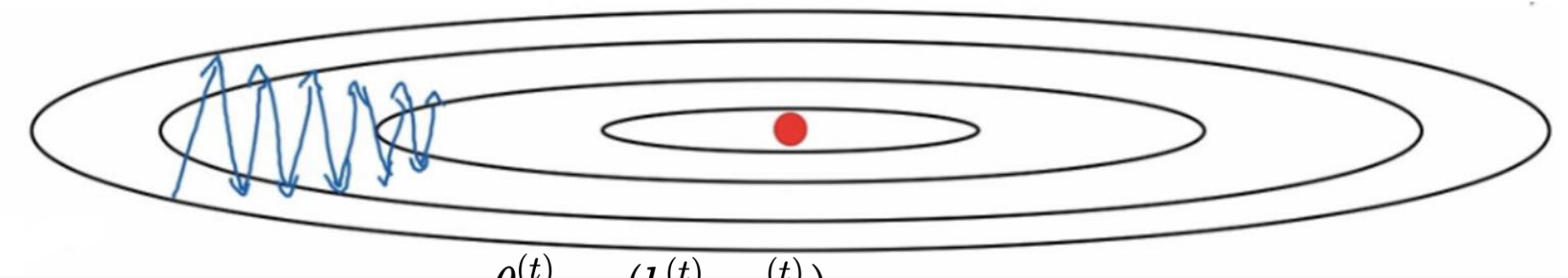
- O mais comum tem sido backprop + momentum
- Existem duas modificações de backprop que também aparecem com frequência:
  - RMSProp
  - ADAM
- Vamos ver cada um deles a seguir.

RMSProp



- Outra tentativa de evitar o zig-zag
- Apropriado quando este zig-zag tem diferentes tamanhos ao longo das diferentes dimensões do vetor de parâmetros
  - Eixo horizontal: parâmetro de bias b
  - Eixo vertical: parâmetro de peso w
- Oscilação excessiva ao longo da direção vertical (w)
- Queremos amenizar esta oscilação vertical e aumentar o tamanho dos passos ao longo de b

RMSProp



$$\theta^{(t)} = (b^{(t)}, w^{(t)})$$

- Vetor de parâmetros
- Vetor gradiente $\nabla \mathcal{L}(\theta^{(t)}) = \left[\frac{\partial \mathcal{L}}{\partial b}, \frac{\partial \mathcal{L}}{\partial w}\right]$
- Variabilidade em cada coordenada

$$S_b^{(t+1)} = \gamma S_b^{(t)} + (1-\gamma)\left[\frac{\partial \mathcal{L}}{\partial b}\right]^2 \quad S_w^{(t+1)} = \gamma S_w^{(t)} + (1-\gamma)\left[\frac{\partial \mathcal{L}}{\partial w}\right]^2$$

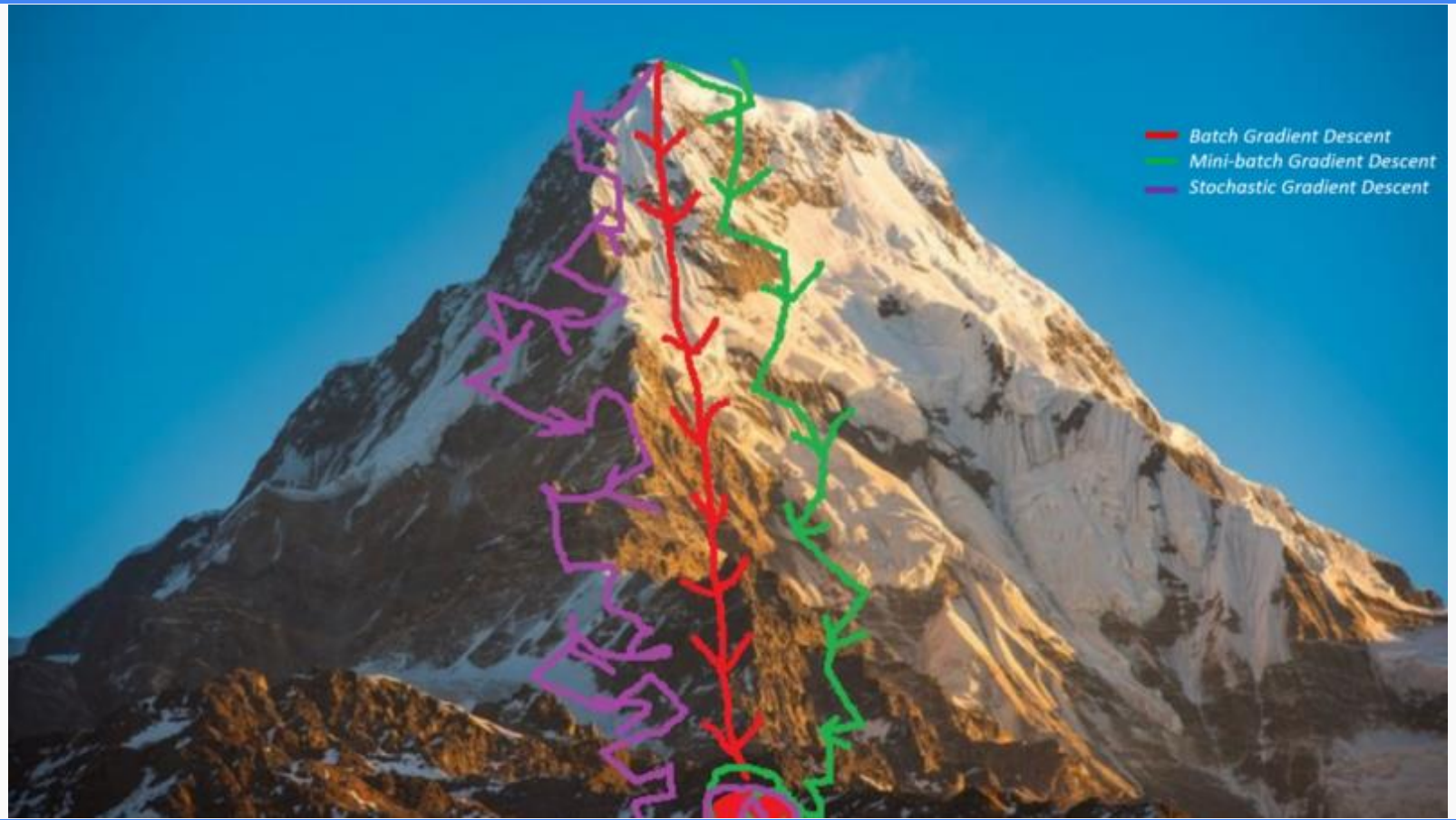
- Atualização

$$w^{(t+1)} = w^{(t)} - \alpha \frac{1}{\sqrt{S_w^{(t+1)}} + \varepsilon} \frac{\partial \mathcal{L}}{\partial w}$$

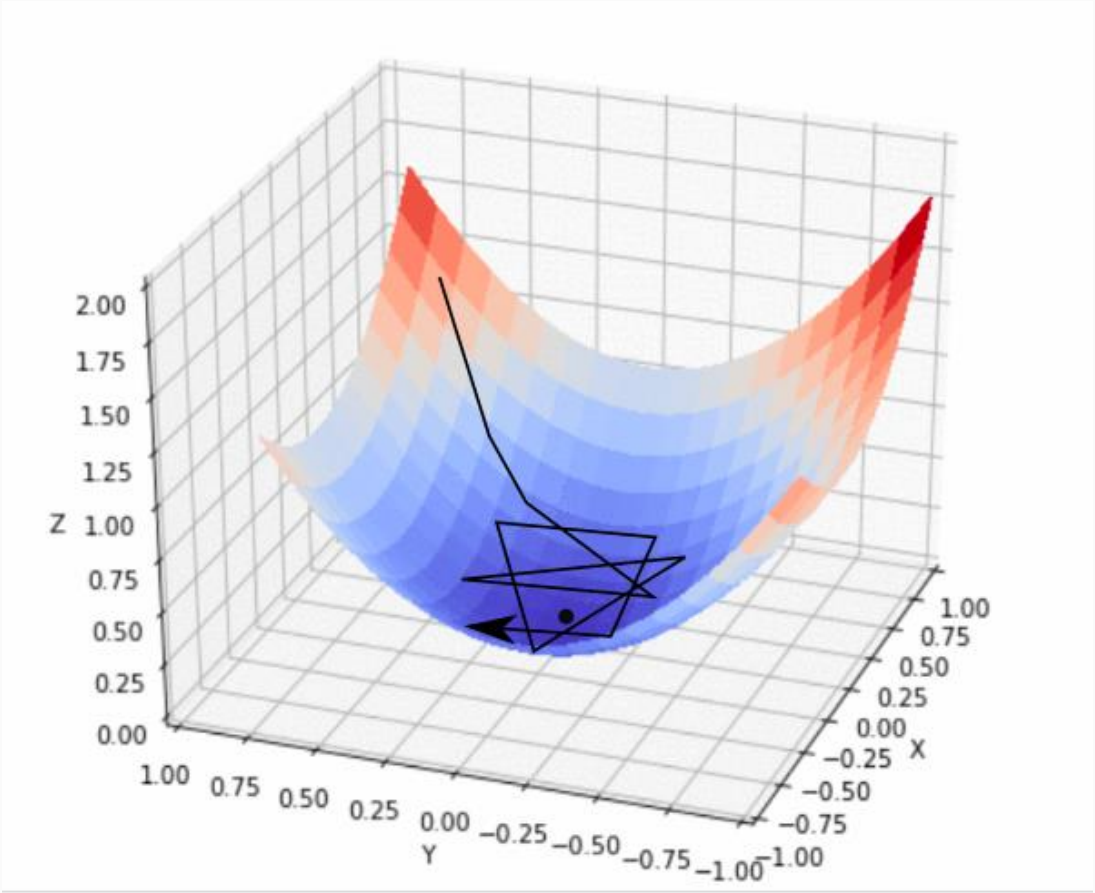
$$b^{(t+1)} = b^{(t)} - \alpha \frac{1}{\sqrt{S_b^{(t+1)}} + \varepsilon} \frac{\partial \mathcal{L}}{\partial b}$$

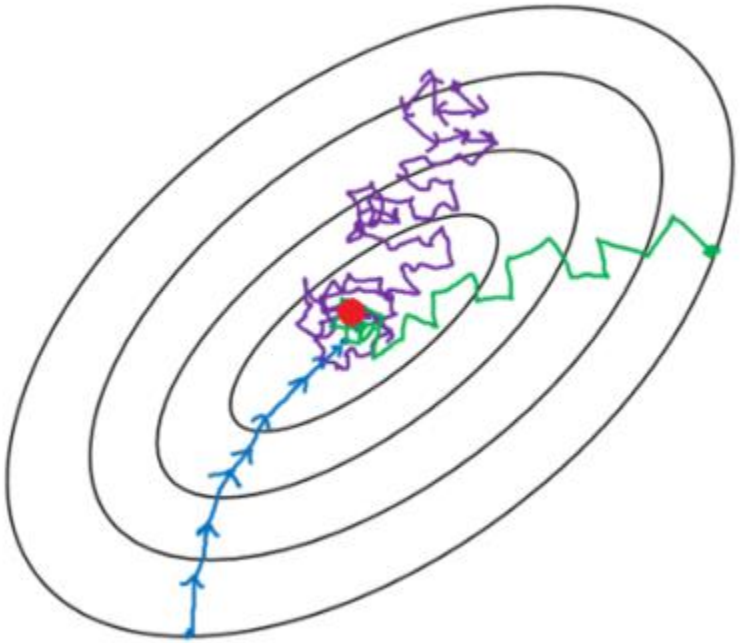
$$\varepsilon = 10^{-6}$$

Stochastic Gradient Descent

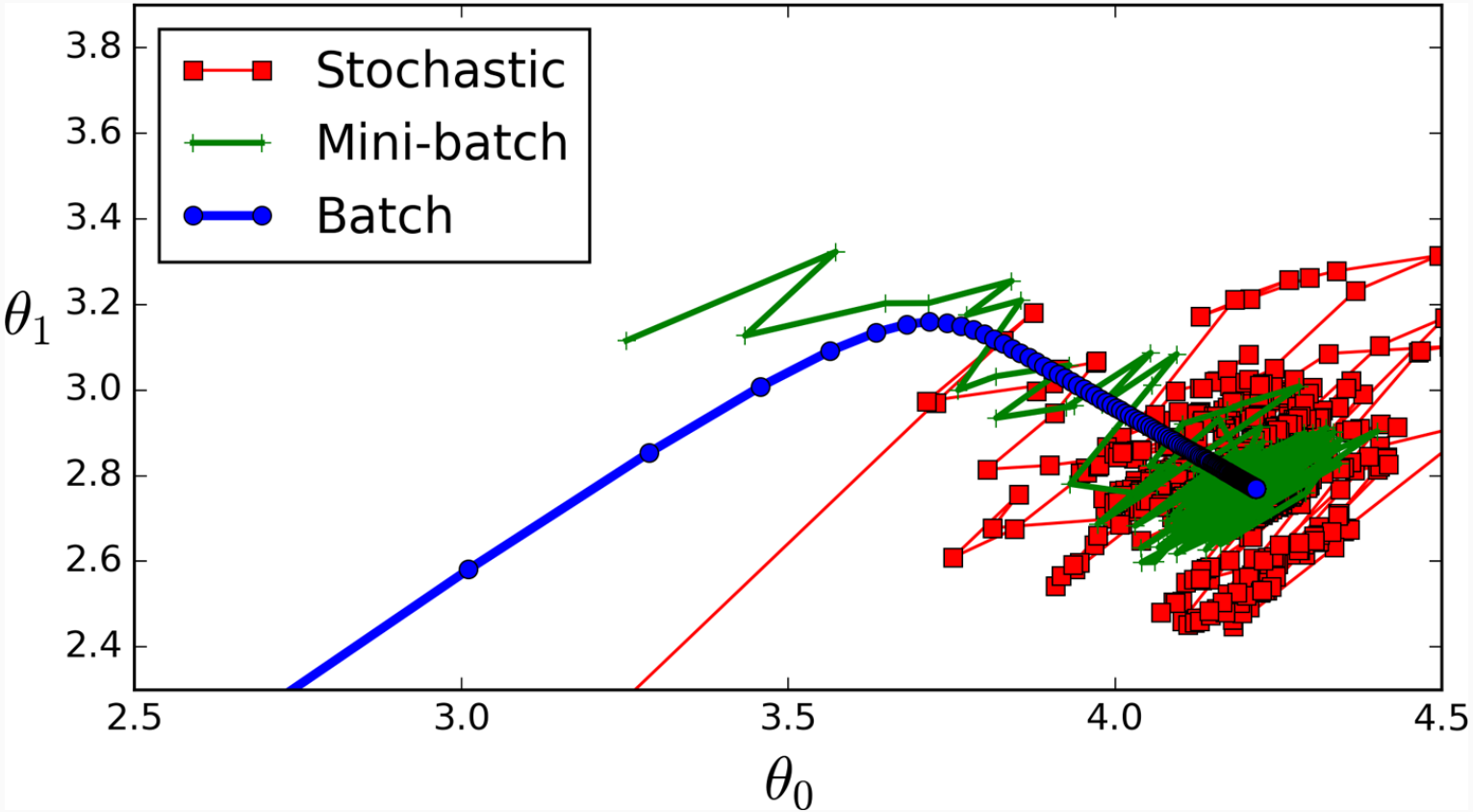


Stochastic Gradient Descent





Batch gradient descent
Mini-batch gradient Descent
Stochastic gradient descent


Stochastic
Mini-batch
Batch
3.8
3.6
3.4
3.2
3.0
2.8
2.6
2.4
θ_1
2.5
3.0
3.5
4.0
4.5
θ_0

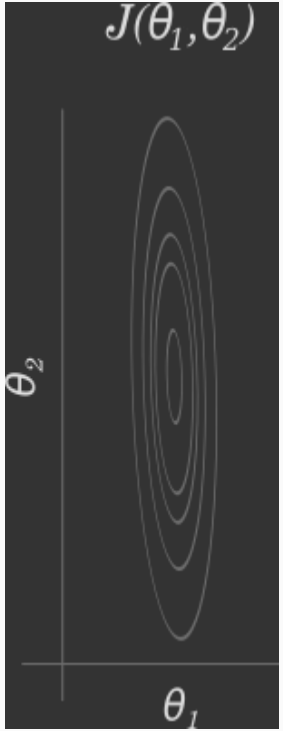
Mini-Batch size

- Se conjunto de dados é pequeno (< 3000), use todos os dados
- Se conjunto de dados não é pequeno, use batch size de acordo com o problema:
  - > 64
  - < 1024
  - Algum valor intermediário entre estes é razoável: 64, 128, 256, 512, 1024
- Importante: escolha os dados do mini-batch de forma aleatória:
  - Não siga uma ordem temporal geográfica ou seguindo alguma característica que possa estar associada com a reposta

Normalizando features

- Features aparecem com ordens de grandeza muito diferentes.
- Por exemplo, idade varia de 0 a 100 anos
- A renda mensal R$ 800 a R$ 100.000 (e mais).
- Usar os dados crus como entrada, não normalizados, pode desacelerar o gradiente descendente:
  - dados crus distorcem a função de custo tornando o ponto mínimo difícil de alcançar.
- Um truque importante em ML: garantir que todos as features estejam em uma escala similar.
- Esta é uma etapa de pré-processamento dos dados.

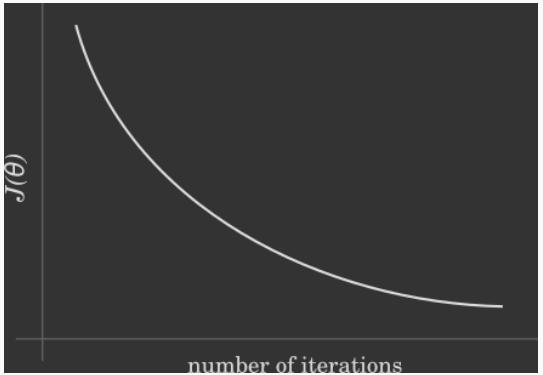
- Suponha que você tenha duas features em indivíduos:
  - x1 como a renda mensal (800 - 100.000);
  - x2 como a idade (0-100).
- Gráfico ao lado mostra a função de custo típica que vai aparecer.
  - uma pequena mudança no coeficiente θ_1 da variável x1 faz o preditor (escore) linear variar muito: $\theta_0 + \theta_1 x_1 + \theta_2 x_2$
  - O escore mudando muito, a função de custo $J(\theta_1, \theta_2)$ também muda muito.
- Já uma pequena mudança no coeficiente θ_2 não afeta muito o escore e nem a função de custo $J(\theta_1, \theta_2)$
- A função de custo é um "monte" muito fino e esticada.
- Gradiente descendente vai oscilar muito para frente e para trás, demorando muito para encontrar o caminho até o ponto mínimo.



- Normalizando as features, terminamos com $J(\theta_1, \theta_2)$ numa forma de "monte" mais esférico
- Gradiente descendente vai ser mais eficiente.
- Como normalizar?
  - min-max
    - por todas as features no intervalo [−1,1].
      - x* ← (x-min(x)) / (max(x) - min(x))
      - Renda* ← (Renda - 800) / (100000 - 800)
  - padronização estatística (ou z-escore):
    - features têm média zero e desvio-padrão 1
      - x* ← (x - mean(x)) / sd(x)
      - Renda* ← (Renda - 2500) / 22400
- Regra geral: quando em dúvida, padronize os dados. Mal não vai fazer e pode ajudar bastante.

Monitore a função de custo $J(\theta_0, \theta_1, \ldots, \theta_p)$

- Verifique se o gradiente descendente está funcionando corretamente.
- Queremos o valor dos THETAs que minimizam a função de custo
- Plotamos a função de custo J versus a iteração do algoritmo para ver se estamos diminuindo J a cada passo.
- O número de iterações no eixo horizontal, a função de custo J na vertical.
    - Em cada iteração, o gradiente descendente produz novos valores de θs
    - Com esses novos valores, avaliamos a função de custo J(θ).
    - Devemos ter uma curva decrescente se o algoritmo se comportar bem
    - Significa que ele está minimizando o valor dos θs corretamente.



Checando convergência com plot de $J(\theta)$

- Plotar $J(\theta)$ também informa se o gradiente descendente convergiu ou não.
- Não existe um único número de iterações até a convergência. Isto varia em função de:
  - qualidade do valor inicial
  - tamanho do problema (número de parâmetros)
  - grau de concentração da superfície $J(\theta)$ em torno do ponto de mínimo
- Em geral, você pode assumir que o algoritmo encontrou um mínimo quando $J(\theta)$ diminui menos que algum valor pequeno ϵ em uma iteração.
  - Escolher um valor adequado ϵ não é uma tarefa fácil.
  - Algumas pessoas tomam ϵ = 10^{−3} e usam um teste de convergência automática:
    - convergiu se $J(\theta)$ diminuir menos que ϵ em uma iteração.
- Pode declarar convergência, pode exigir também uma diferença relativa pequena para valores sucessivos de $J(\theta)$

Escolha bem a taxa de aprendizagem alpha

- O plot de J(θ) pode ficar estranho:
  - J(θ) crescendo
  - J(θ) diminuindo muuuuuuito lentamente
  - J(θ) oscilando: subindo e descendo, sem diminuir de forma muito consistente
- Uma solução é trocar a learning rate α.
- É provado matematicamente que, para α suficientemente pequeno, J(θ) diminui em cada iteração.
- Por outro lado, se α é muito pequeno, gradient descent pode demorar a convergir (pois theta quase não muda de iteração para iteração).
- Regra geral: tentar um intervalo de valores α.
- Comece com α = 0,001 e observe o gráfico J (θ). Diminui de forma adequada e rápida? Done.
- Caso contrário, mude para α = 0,01 (escala logarítmica) e repita até que o algoritmo funcione bem.