

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



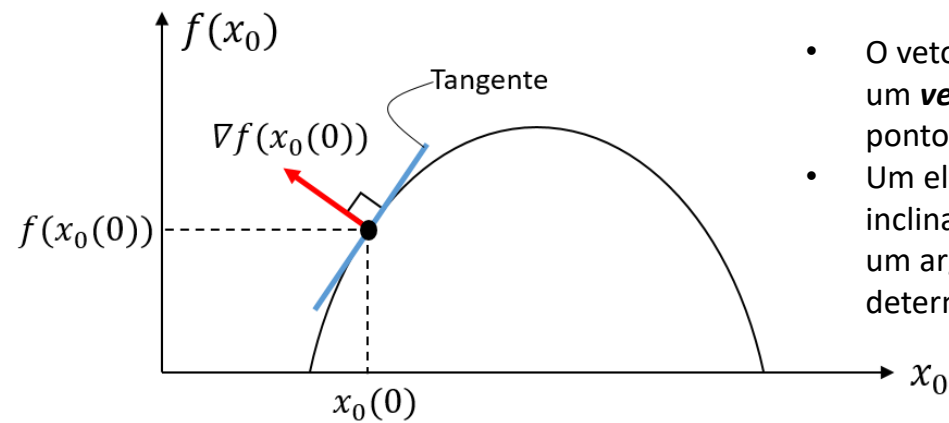
***Inatel***

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

# Recapitulando

- Vimos a ***motivação*** por trás da ***regressão linear***: encontrar funções que aproximem o fenômeno (ou modelo) gerador por trás das observações ruidosas.
- Definimos o ***problema matematicamente***.
- Vimos como resolver o problema da regressão, i.e., ***encontrar os pesos do modelo, através da equação normal e visualmente***.
- Aprendemos o que é uma ***superfície de erro***.
- Discutimos algumas ***desvantagens*** (e.g. ***complexidade, regressão não-linear***) da equação normal e vislumbramos uma solução para essas desvantagens, a qual discutiremos a seguir.

# Vetor Gradiente



- O vetor gradiente em um ponto específico é um **vetor ortogonal** à reta tangente àquele ponto.
- Um elemento do vetor gradiente dá a inclinação de uma reta tangente em relação a um argumento específico da função em um determinado ponto.

- Vocês se lembram das aulas de cálculo vetorial, onde vocês aprenderam sobre o **vetor gradiente**?
  - **Vetor gradiente** nos dá a **direção** e a **magnitude** da maior taxa de variação de uma função  $f(x)$  em um determinado ponto.

- O **vetor gradiente** de uma função  $f(x_0, x_1, \dots, x_K)$  com  $K$  argumentos é definido pela derivada parcial em relação a cada um de seus argumentos  $x_k, k = 0, \dots, K$ :

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

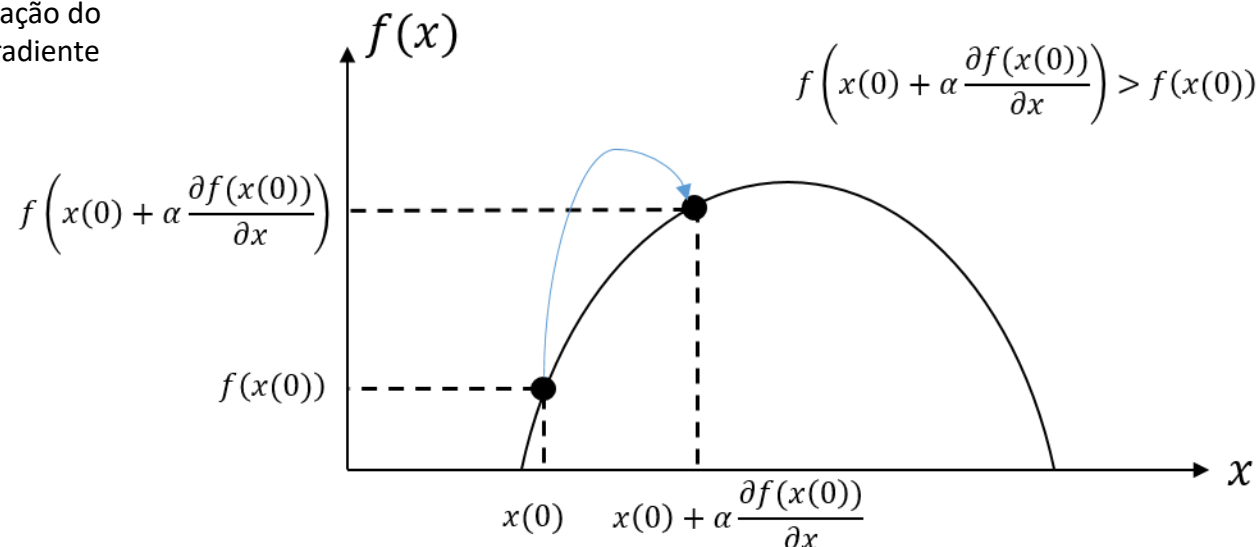
- Cada **elemento** do vetor gradiente nos dá a **taxa de variação da função em relação àquele argumento em um determinado ponto** (pode ser interpretado como a inclinação de uma a reta tangente ao ponto).
- Se o **vetor gradiente** de uma função **em um determinado ponto é igual a zero**, significa que **a função não varia em nenhuma direção**. Isso indica um extremo da função (máximo ou mínimo).

# Vetor Gradiente

- O vetor gradiente aponta para a **direção** em que, ao se **mover** a partir de um **ponto específico**, a **função  $f(x)$  cresce mais rapidamente**.
- Se imaginem parados em um 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 qual direção devemos caminhar para aumentar o valor de  $f$  mais rapidamente, ou seja

$$f\left(x_0(0) + \alpha \frac{\partial f(x_0, x_1, \dots, x_K)}{\partial x_0}, \dots, x_K(0) + \alpha \frac{\partial f(x_0, x_1, \dots, x_K)}{\partial x_K}\right) > f(x_0(0), \dots, x_K(0)).$$

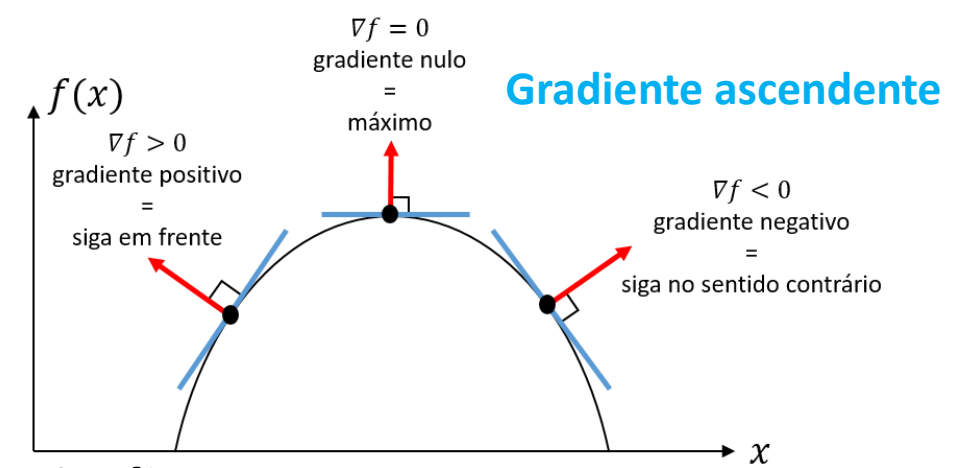
Fração do gradiente



**OBS.:**

- Se, a cada novo ponto, calcularmos o vetor gradiente e adicionarmos uma fração dele ao ponto atual, teremos um novo ponto que leva a um valor da função maior do que o valor anterior.
- Portanto, podemos criar um procedimento que vá iterativamente em direção ao máximo da função.

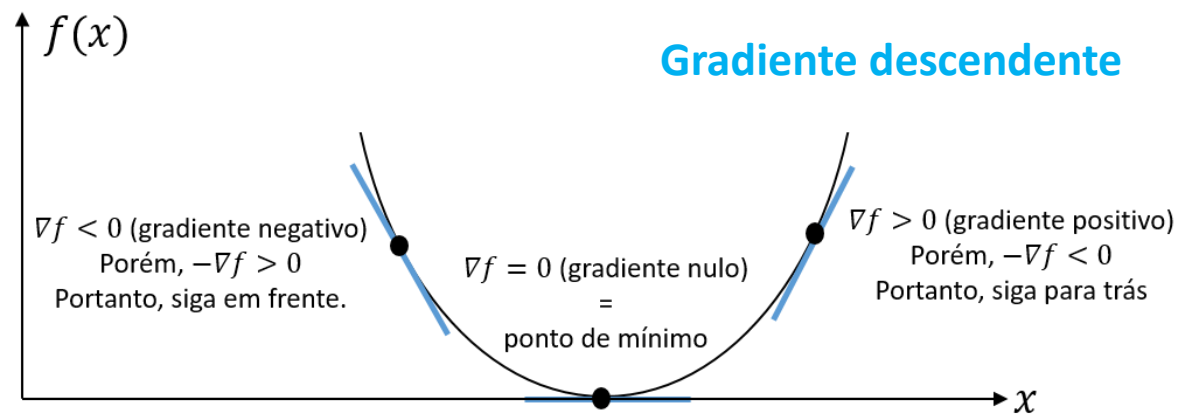
# Gradiente Ascendente



- A **derivada parcial** (i.e., elemento de  $\nabla f(x)$ ) dá a **inclinação** de uma **reta tangente** em relação a um argumento específico de  $f(x)$  em um determinado **ponto**.
- Assim, neste **ponto**, cada elemento do **vetor gradiente** com valor:
  - + (inclinação positiva) indica que o ponto de máximo está à frente do ponto.
  - - (inclinação negativa) indica que o ponto de máximo está atrás do ponto.
  - 0 (inclinação nula) indica que ponto de máximo foi encontrado.
- Portanto, seguindo na direção indicada pelo **vetor gradiente**, chegamos ao ponto de máximo da função,  $f(x_0, x_1, \dots, x_K)$ .
- Assim, um algoritmo de otimização **iterativo** que siga a direção indicada pelo **vetor gradiente** para encontrar o **ponto de máximo** de uma função  $f(x_0, x_1, \dots, x_K)$  é conhecido como **gradiente ascendente**.
- A cada **iteração**,  $l$ , calcula-se o **vetor gradiente** da função  $f(x)$  num ponto específico,  $x(l)$ , e atualiza-se os valores dos argumentos da função de tal forma, que a cada **iteração** se tenha:

$$f(x(l+1)) = f(x(l) + \alpha \nabla f(x(l))) > f(x(l)), l \geq 0.$$

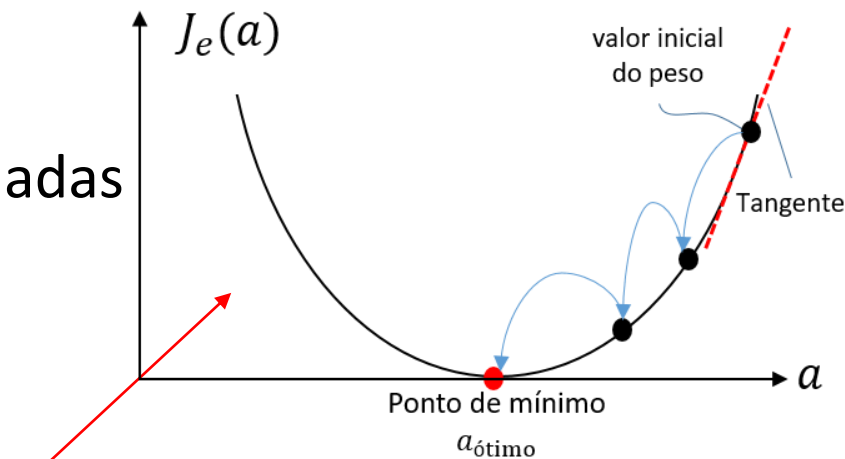
# Gradiente Descendente



- Mas e se formos na direção contrária a da máxima taxa de crescimento, dada pelo **vetor gradiente**,  $\nabla f(x_0, x_1, \dots, x_K)$ , ou seja  $-\nabla f(x_0, x_1, \dots, x_K)$ ?
  - Neste caso, iremos na direção de **decréscimo** mais rápido da função,  $f(x_0, x_1, \dots, x_K)$ .
- Portanto, um algoritmo de otimização **iterativo** que siga a direção contrária a indicada pelo **vetor gradiente** para encontrar o **ponto de mínimo** de uma função  $f(x_0, x_1, \dots, x_K)$  é conhecido como **gradiente descendente**.
- A cada **iteração**,  $l$ , calcula-se o **vetor gradiente** da função  $f(x)$  num ponto específico,  $x(l)$ , e atualiza-se os valores dos argumentos da função de tal forma, que a cada **iteração**, se tenha o valor de  $f(x)$  **menor** do que o anterior:
$$f(x(l+1)) = f(x(l) - \alpha \nabla f(x(l))) < f(x(l)), l \geq 0.$$
- Nesta disciplina, como queremos minimizar o erro, iremos focar neste algoritmo.

# Características do Gradiente Descendente

- Algoritmo de **otimização iterativo** e **genérico**: encontra soluções ótimas para uma ampla gama de problemas.
  - Por exemplo, é utilizado em vários problemas de aprendizado de máquina e otimização.
- Escalona melhor do que o método da **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**).
- Pode ser usado com modelos não-lineares.
- O único requisito é que a **função de erro** seja **diferenciável**.
- Quando aplicado a problemas de **regressão**, a ideia geral é atualizar os pesos,  **$a$** , **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**.



A cada nova iteração de atualização (seta azul), o peso se aproxima de seu valor ótimo, consequentemente, minimizando o erro.

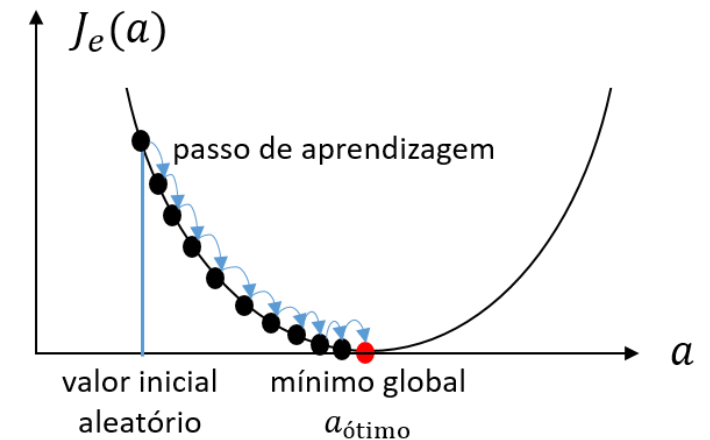
# O Algoritmo do Gradiente do Descendente (GD)

- O algoritmo inicializa os pesos,  $\mathbf{a}$ , em um ponto aleatório do **espaço de pesos** e, então, aplica a **regra de atualização dos pesos** até que o algoritmo convirja (e.g., erro pequeno entre duas iterações subsequentes) ou o número máximo de iterações seja atingido.

$\mathbf{a} \leftarrow$  inicializa em um ponto qualquer do espaço de pesos  
loop até convergir ou atingir o número máximo de iterações do

$$\mathbf{a} \leftarrow \mathbf{a} - \alpha \frac{\partial J_e(\mathbf{a})}{\partial \mathbf{a}} \text{ (regra de atualização dos pesos)}$$

Os pesos são atualizados na direção oposta a do vetor gradiente.



onde  $\alpha > 0$  é a **passo de aprendizagem** e  $\frac{\partial J_e(\mathbf{a})}{\partial \mathbf{a}}$  é o **vetor gradiente**,  $\nabla J_e(\mathbf{a})$ , da **função de erro**, ou seja, a derivada parcial da função em relação ao vetor de pesos,  $\mathbf{a}$ .

- O **passo de aprendizagem** dita o tamanho dos deslocamentos dados na direção oposta a do **gradiente**.
- O **passo de aprendizagem** pode ser constante ou pode decair com o tempo à medida que o processo de aprendizado prossegue.
- Na sequência, veremos como encontrar o **vetor gradiente** da **função de erro** e como implementar o algoritmo do **gradiente descendente**.



# Exemplo

[Exemplo: exemplo\\_regressao\\_linear\\_gradiente\\_descendente.ipynb](#)

- Usaremos uma **função hipótese** com 2 pesos,  $a_1$  e  $a_2$

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

- A função de erro é dada por

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

- Cada elemento do vetor gradiente é dado por

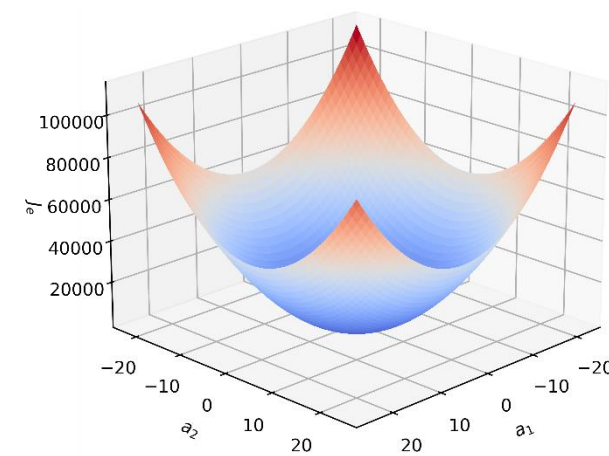
$$\frac{\partial J_e(\mathbf{a})}{\partial a_k} = -\frac{2}{N} \sum_{n=0}^{N-1} [y(n) - (a_1 x_1(n) + a_2 x_2(n))] x_k(n), k = 1, 2$$

- A **equação de atualização** dos pesos  $a_k$ ,  $k = 1$  e  $2$  é dada por

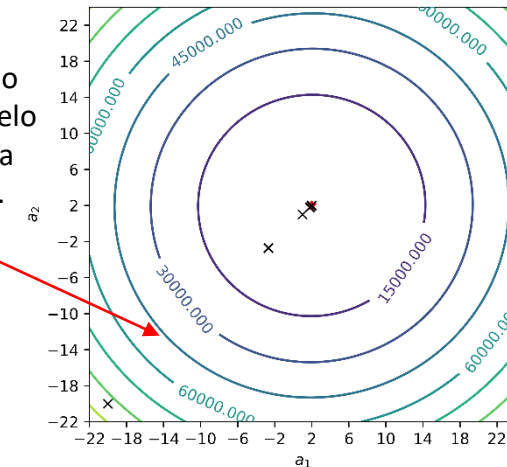
$$a_k = a_k - \alpha \frac{\partial J_e(\mathbf{a})}{\partial a_k}$$

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

- Por ser constante, o termo  $2/N$  pode ser absorvido por  $\alpha$ .
- Forma matricial da equação de atualização:  $\mathbf{a} = \mathbf{a} - \alpha \mathbf{X}^T (\mathbf{y} - \hat{\mathbf{y}})$



Superfície de contorno com o caminho feito pelo algoritmo até a convergência.



# Versões do Gradiente Descendente

- Existem três versões diferentes para a implementação do algoritmo do gradiente descendente:
  - Batelada;
  - Estocástico;
  - Mini-Batch.

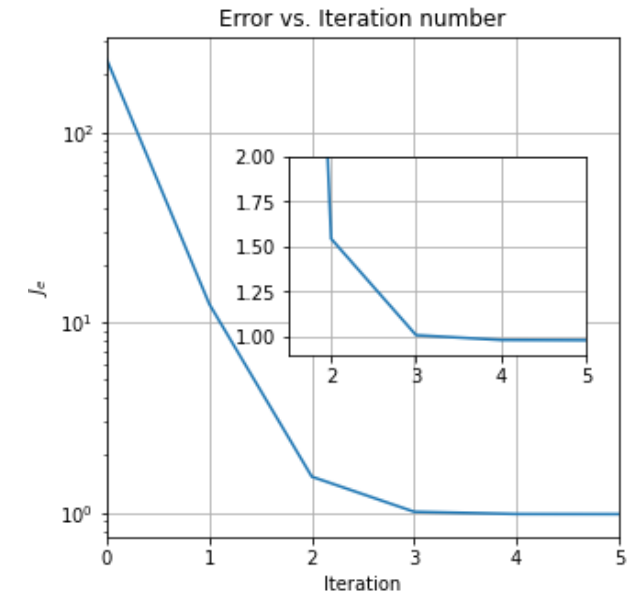
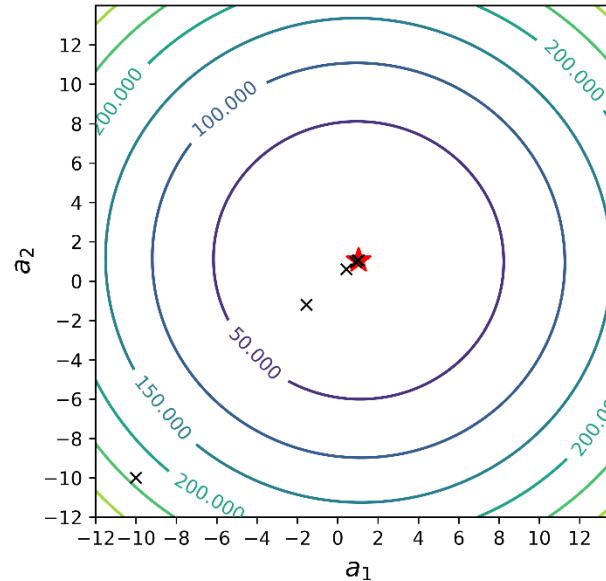
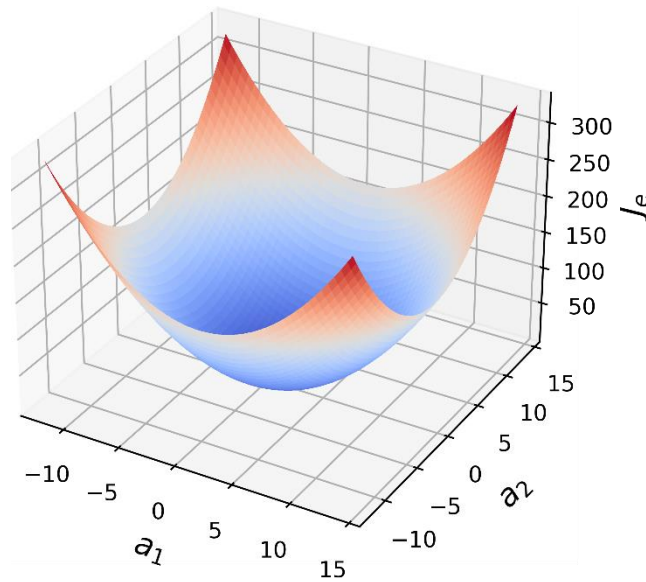
# Versões do Gradiente Descendente

- **Batelada** (do inglês ***batch***): a cada época do algoritmo, ***todos*** os exemplos de treinamento são considerados no processo de treinamento do modelo. Esta versão foi a utilizada no exemplo anterior.

$$a_k = a_k + \alpha \sum_{n=0}^{N-1} [y(n) - h(\mathbf{x}(n))] x_k(n), \quad k = 1, \dots, K$$

- **Características:**
  - Utilizado quando se possui previamente todos os atributos,  $\mathbf{x}$ , e rótulos,  $y$ , de treinamento.
  - **Convergência garantida**, dado que o passo de aprendizagem tenha o tamanho apropriado e se espere tempo suficiente.
  - **Convergência pode ser bem lenta**, dado que o modelo é apresentado a todos os exemplos a cada época.
  - Se o conjunto de treinamento for muito grande, pode ser impossível treinar o modelo, pois ele consome muitos recursos computacionais (CPU e memória).

# Características do GD em Batelada



- Segue diretamente para o mínimo global.
- Atinge o mínimo global em aproximadamente 3 épocas.
- Nesse caso específico, segue uma linha reta entre  $a_1$  e  $a_2$  pois a taxa de decrescimento da superfície de erro é igual para os dois pesos (contornos são circulares).
- Não fica “*oscilando*” em torno do mínimo após alcançá-lo, pois o vetor gradiente neste ponto é praticamente nulo.

# Versões do Gradiente Descendente

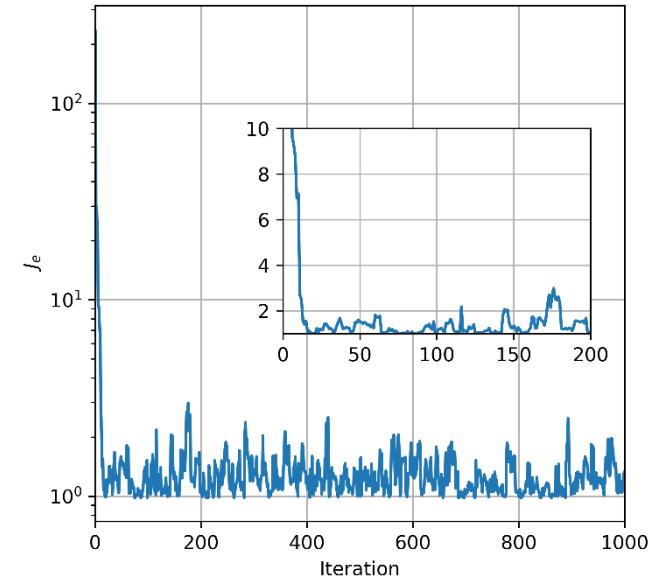
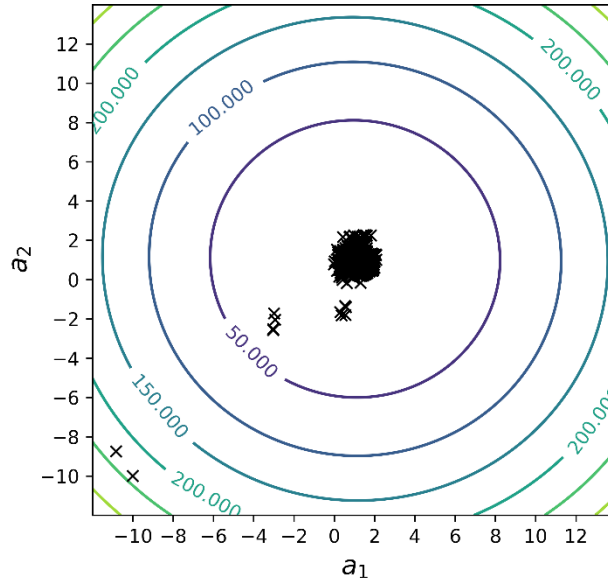
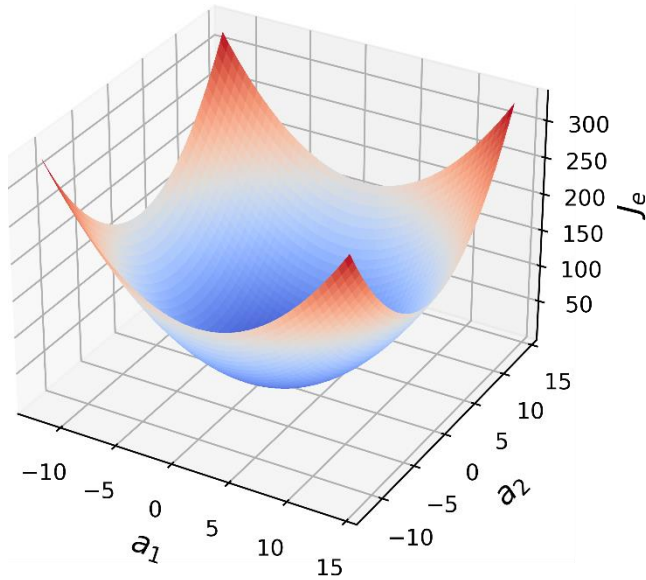
- **Gradiente Descendente Estocástico (GDE)**: também conhecido como ***online*** ou ***incremental*** (exemplo-a-exemplo). Com esta versão, os pesos do modelo são atualizados a cada novo exemplo de treinamento.

$$a_k = a_k + \alpha [y(n) - h(\mathbf{x}(n))] x_k(n), \quad k = 1, \dots, K$$

- **Características:**

- ***Aproxima o gradiente*** através de uma ***estimativa estocástica (aleatória)***, ou seja, aproxima o gradiente usando apenas um exemplo de treinamento.
- Essa ***aproximação é ruidosa***, o que faz com que a ***aproximação do vetor gradiente tenha direções divergentes a cada iteração***.
- Utilizado quando os ***atributos e rótulos*** são ***obtidos sequencialmente*** (e.g., sensores).
- Ou quando o ***conjunto de treinamento é muito grande*** (toma-se amostras aleatoriamente).
- ***Computacionalmente mais rápido e menos custoso em termos de CPU e memória*** que o GD em batelada.
- ***Convergência não é garantida*** com um passo de aprendizagem fixo. O algoritmo pode ***oscilar*** em torno do mínimo ***sem nunca convergir*** para o valor ótimo.
- Esquemas de variação do passo de aprendizagem podem ajudar a garantir a convergência.

# Características do GD Estocástico



- Devido à sua natureza aleatória, ***não apresenta um caminho regular para o mínimo***, mudando de direção várias vezes.
- Por aproximar o gradiente com apenas um exemplo, as ***derivadas parciais são “ruidosas”***.
- Por serem ruidosas, o algoritmo não converge suavemente para o mínimo: “*oscila*” em torno dele.
- Quando o treinamento termina, os valores finais dos pesos são bons, mas podem não ser ótimos.
- A convergência ocorre apenas na média.
- Tempo de treinamento é menor: com apenas uma época o algoritmo já se aproxima do ponto ótimo.
- Necessita de um esquema de ajuste do passo de aprendizagem,  $\alpha$ , para ficar mais “*comportado*”.

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

# 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 aleatórios (i.e., par atributo/rótulo), onde os pesos do modelo são ajustados a cada mini-batch.

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

onde  $MB$  é o tamanho do mini-batch.

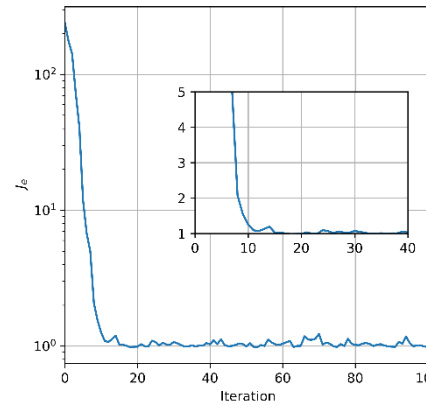
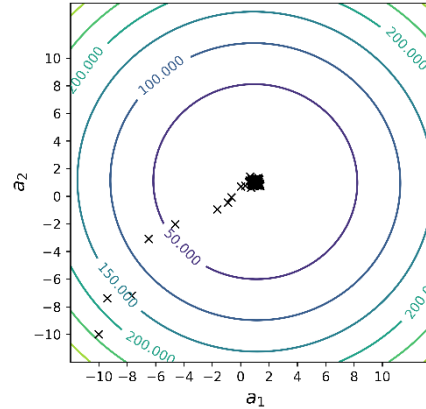
## Características:

- Pode ser visto como uma **generalização** das 2 versões anteriores:
  - Caso  $MB = N$ , então ele se torna o GD em batelada.
  - Caso  $MB = 1$ , então ele se torna o GD estocástico.
- Computacionalmente mais rápido do que o GD em batelada, mas mais lento do que o GD estocástico.
- Convergência depende do tamanho do mini-batch.
- Pode usar esquemas de variação do passo de aprendizagem para melhorar a convergência caso o mini-batch seja muito pequeno.

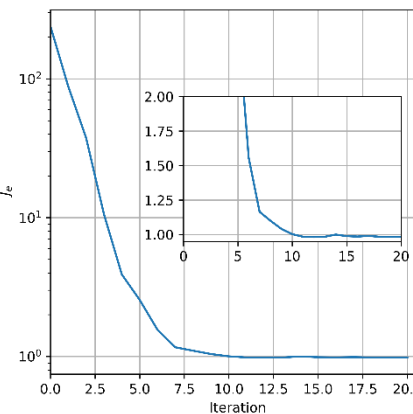
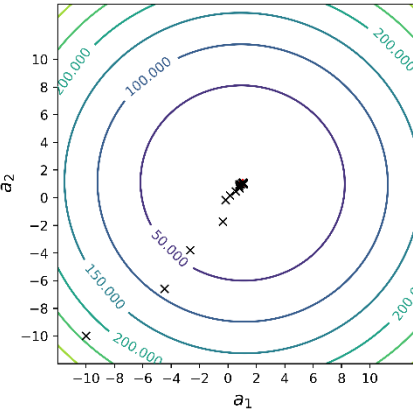
# Características do GD com Mini-Batch

- **Progresso menos irregular** do que com o GDE, especialmente com mini-batches maiores.
- Como resultado, essa versão **oscila menos ao redor do mínimo global** do que o GDE.
- Tem **comportamento mais próximo do GD em batelada** para mini-batches maiores.
- **Oscilação** em torno do mínimo **diminui conforme o tamanho do mini-batch aumenta**.
- Esquema de redução de  $\alpha$  pode balancear **rapidez** e **convergência**.

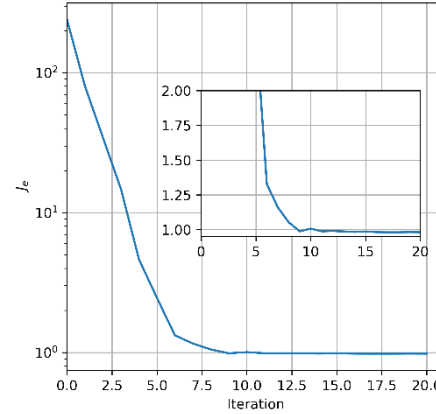
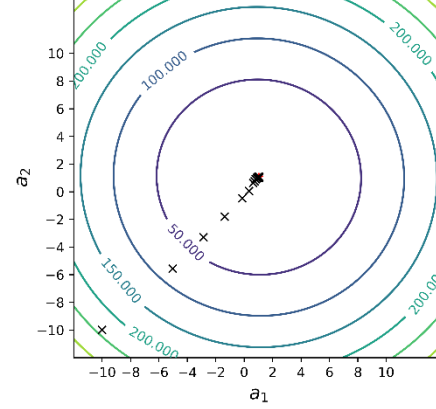
Tamanho do mini-batch: 10



Tamanho do mini-batch: 50



Tamanho do mini-batch: 100



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



# Tarefas

- **Quiz:** “*T319 - Quiz - Regressão: Parte II*” que se encontra no MS Teams.
- **Exercício Prático:** [Laboratório #3](#).
  - Pode ser acessado através do link acima (Google Colab) ou no GitHub.
  - Vídeo explicando o laboratório: Arquivos -> Material de Aula -> Laboratório #3
  - Se atentem aos prazos de entrega.
  - [Instruções para resolução e entrega dos laboratórios](#).
  - **Laboratórios podem ser resolvidos em grupo, mas as entregas devem ser individuais.**

Obrigado!

# Encontrando o vetor gradiente

**Função hipótese** com 2 pesos,  $a_1$  e  $a_2$

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

A função de erro é dada por

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

Cada elemento do vetor gradiente é dado por

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

Operação da  
derivada parcial  
é distributiva.

