

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



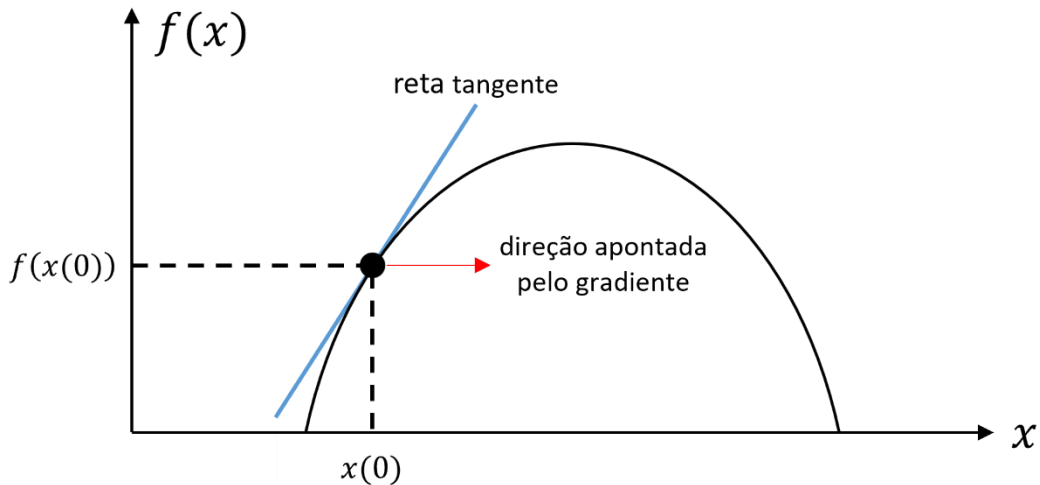
***Inatel***

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

# Recapitulando

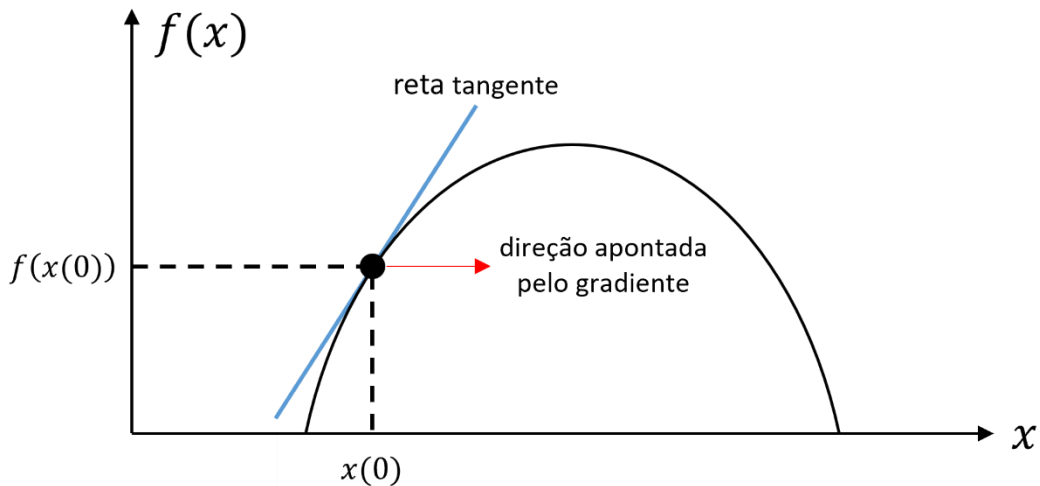
- Vimos a **motivação** por trás da **regressão linear**: encontrar funções que **aproximem o comportamento** de um conjunto de amostras (em geral ruidosas).
- Definimos o **problema matematicamente**.
- Vimos como resolver o problema da regressão, i.e., **encontrar os pesos do modelo, através da equação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

# Vetor gradiente



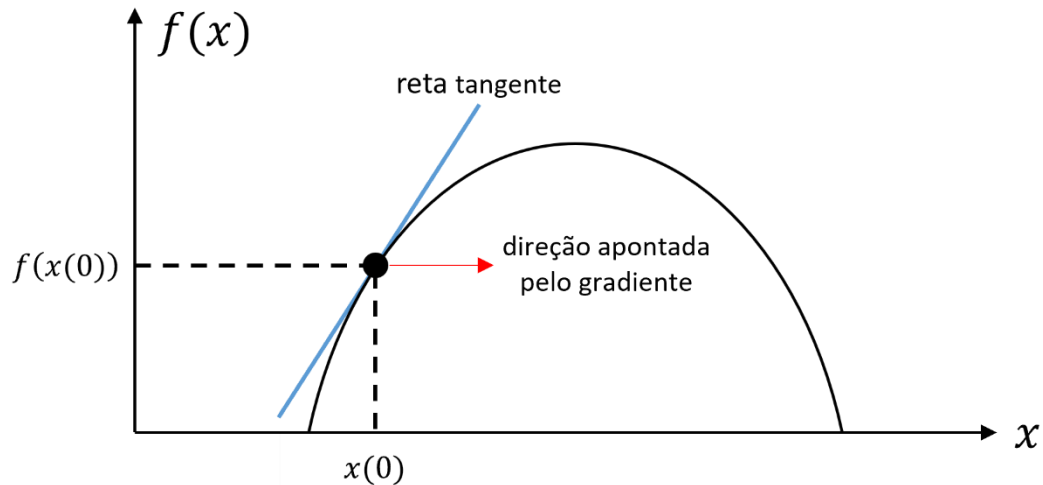
- Vocês se lembram das aulas de cálculo vetorial, onde vocês aprenderam sobre o **vetor gradiente**?
- Qual **informação** ele nos dá sobre uma função?

# Vetor gradiente



- O vetor gradiente aponta na **direção** em que a função  $f(x)$  **cresce mais rapidamente a partir do ponto em que é avaliado**.
- A **magnitude** do vetor gradiente indica a **taxa de crescimento da função** nessa direção.
  - Quanto maior a magnitude, maior a taxa de crescimento naquela direção.
- Ele diz para que “**lado**” (**aumentar ou diminuir**) os valores dos argumentos,  $x$ , devem ir para que o **valor de  $f(x)$  seja maior do que o atual**.

# Vetor gradiente



**Obs.:** No caso da função ter apenas um argumento,  $f(x)$ , o vetor gradiente dá a inclinação de uma **reta** tangente ao ponto onde o vetor é calculado.

- O vetor gradiente pode ser também interpretado como a **inclinação de um plano tangente à função** no ponto onde ele é calculado.
  - Quanto **maior o valor absoluto** do gradiente, **mais inclinado é o plano tangente** naquele ponto.
  - Portanto, **um vetor gradiente igual a 0 indica inclinação nula**.
  - Ou seja, **a função não varia mais em nenhuma direção**.
  - Onde isso ocorre? Nos **extremos da função**, ou seja, em seus **pontos de máximo e de mínimo**.

# Vetor gradiente

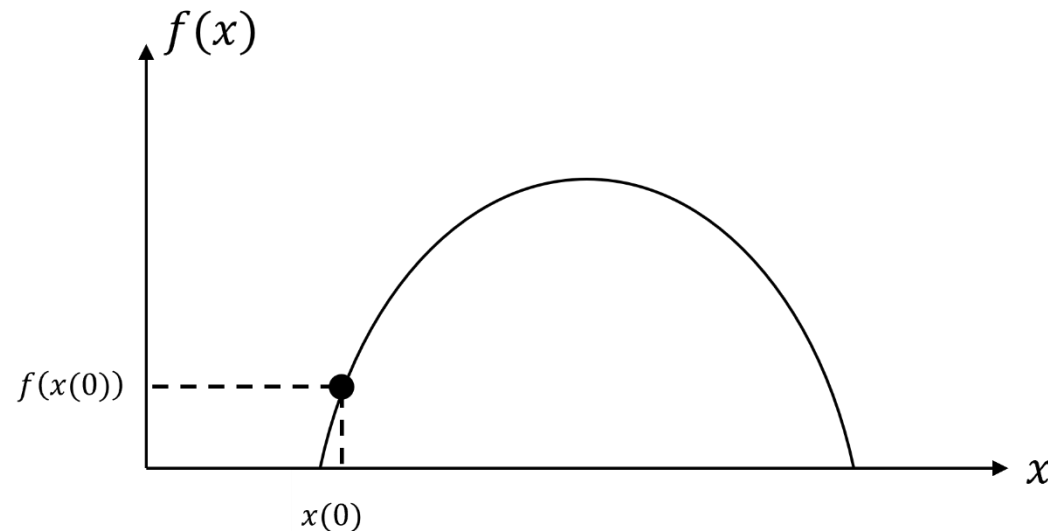
- O **vetor gradiente** de uma função com  $K$  argumentos,  $f(x_1, x_2, \dots, x_K)$ , é definido pela **derivada parcial em relação a cada um de seus argumentos**  $x_k, k = 1, \dots, K$ :

$$\begin{aligned} & \nabla f(x_1, x_2, \dots, x_K) \\ &= \left[ \frac{\partial f(x_1, x_2, \dots, x_K)}{\partial x_1} \quad \frac{\partial f(x_1, x_2, \dots, x_K)}{\partial x_2} \quad \dots \quad \frac{\partial f(x_1, x_2, \dots, x_K)}{\partial x_K} \right]^T. \end{aligned}$$

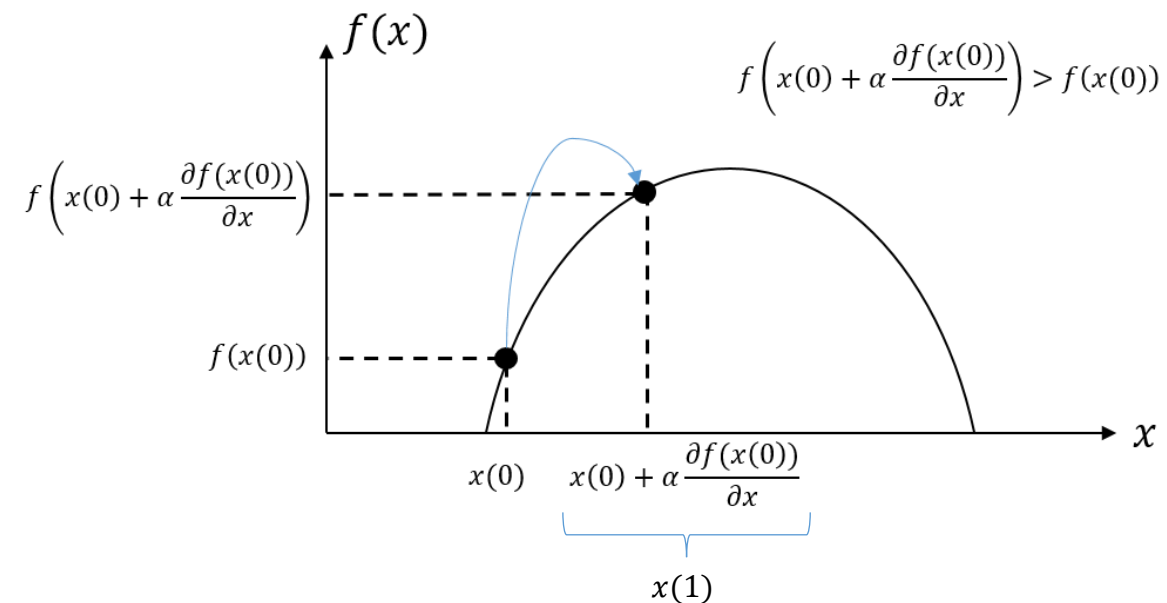
- Notem que o vetor gradiente é representado pelo símbolo *Nabla*,  $\nabla$ , e é definido como um **vetor coluna**, com número de elementos igual ao número de argumentos da função.
- **OBS.:** Na sequência, sem perda de generalidade, nós vamos assumir uma função com apenas um argumento,  $f(x)$ .

# O vetor gradiente indica o caminho para o máximo da função

- Imaginem o ponto inicial,  $x(0)$ , com valor  $f(x(0))$  na figura abaixo.
- Se quisermos que o valor de  $f(x)$  aumente, devemos aumentar ou diminuir o valor de  $x(0)$ ?
- Ou seja, qual **direção** devemos seguir para **maximizar** o valor de  $f(x)$ ?



# O vetor gradiente indica o caminho para o máximo da função

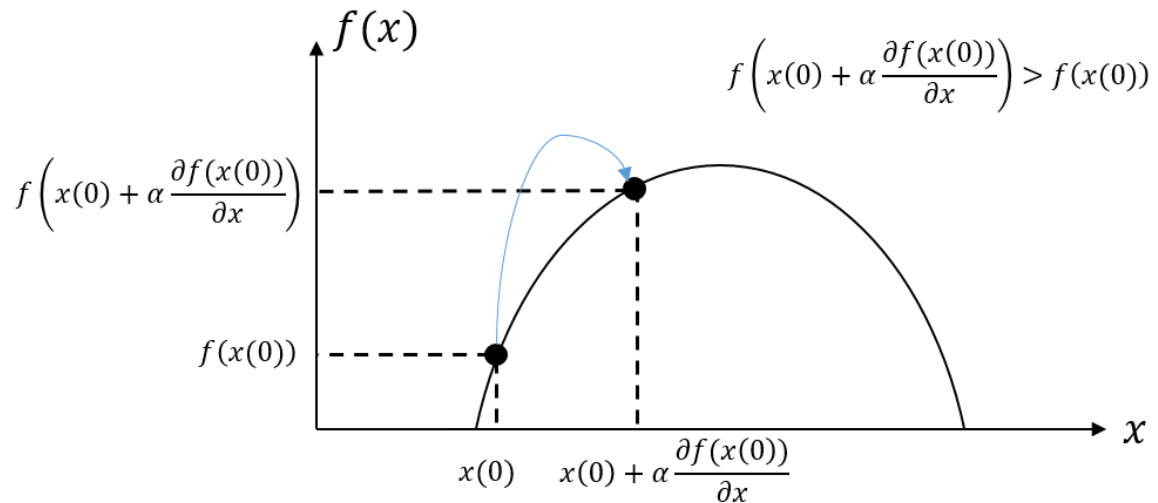


- O vetor gradiente calculado no ponto  $x(0)$ ,  $\nabla f(x(0))$ , diz **em qual direção** devemos caminhar para **aumentar o valor da função**  $f(x)$  mais rapidamente.
- Se **adicionarmos** uma **porcentagem**,  $\alpha$ , do gradiente ao valor de  $x(0)$ , teremos que o **novo ponto**,  $x(1)$ , terá um valor de  $f(x)$  **maior do que o anterior**, ou seja

$$f\left(\underbrace{x(0) + \alpha \frac{\partial f(x(0))}{\partial x}}_{x(1)}\right) > f(x(0)).$$

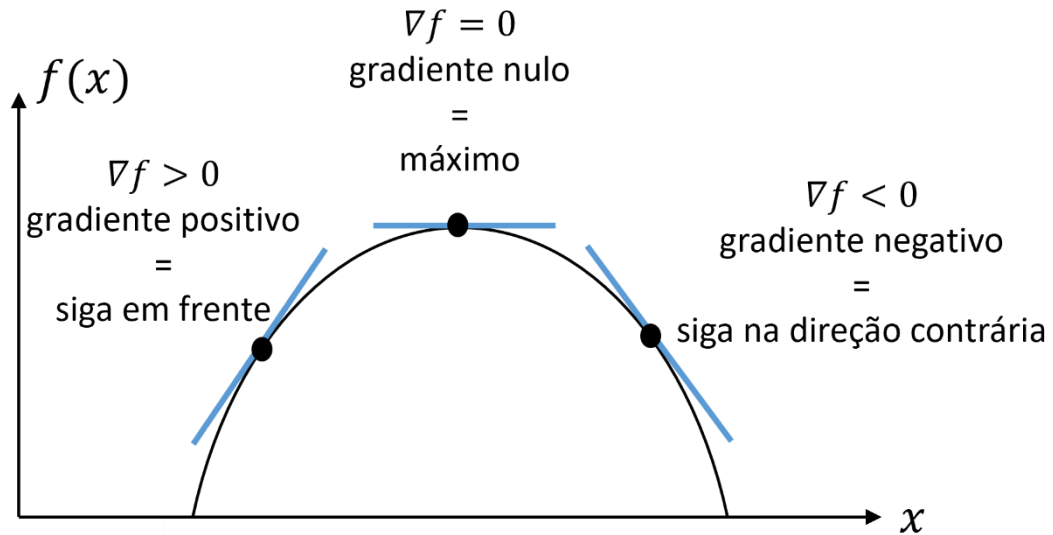


# O vetor gradiente indica o caminho para o máximo da função



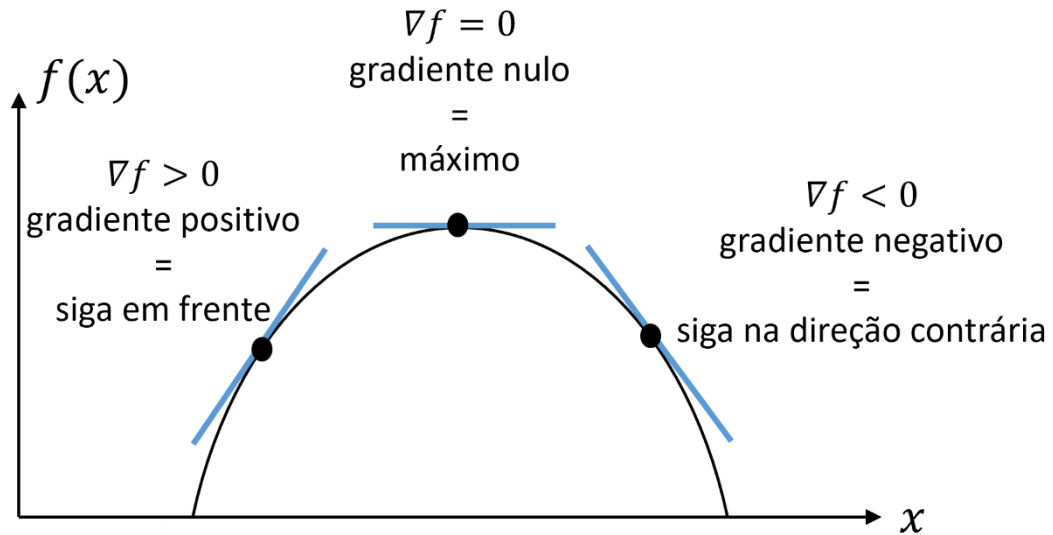
- Se, a cada **ponto atual**, nós calcularmos o vetor gradiente e adicionarmos uma **porcentagem** dele ao ponto, teremos um **novo ponto** que leva a um **valor da função maior do que o valor anterior**.
- Portanto, podemos criar um **procedimento** que vá **iterativamente caminhando** em **direção ao ponto de máximo da função**.
- Vamos entender como isso pode ser feito.

# Algoritmo do gradiente ascendente



- Se o vetor gradiente de  $f(x)$  em um **ponto**  $x(n)$  qualquer dá a **inclinação da reta tangente à função naquele ponto**.
- Então, nesse **ponto**, um valor de gradiente:
  - + (reta com inclinação positiva) indica que o **ponto de máximo está à frente do ponto atual**.
  - - (reta com inclinação negativa) indica que o **ponto de máximo está atrás do ponto atual**.
  - 0 (reta com inclinação nula) indica que **ponto de máximo foi encontrado**.

# Algoritmo do gradiente ascendente



- Portanto, *seguindo na direção* indicada pelo *vetor gradiente*, *chegamos* ao *ponto de máximo da função*.
- Um algoritmo *iterativo* de otimização que *siga a direção* indicada pelo *vetor gradiente* para encontrar o *ponto de máximo* de uma função é conhecido como *gradiente ascendente*.
- Mas como ele funciona?

# Algoritmo do gradiente ascendente

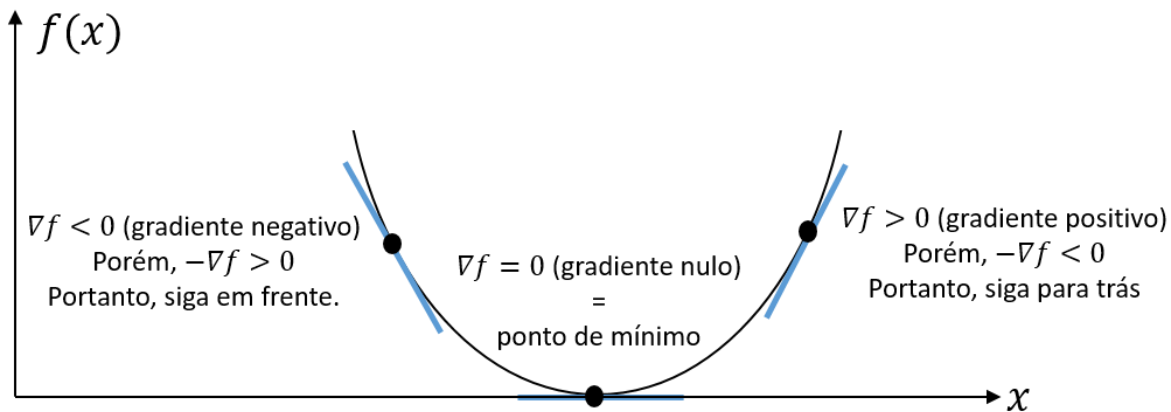
- Inicializa-se o argumento  $x(0)$  com um valor arbitrário, em geral, um valor **aleatório**.
- A cada **iteração**,  $i$ , calcula-se o **vetor gradiente** da função  $f(x)$  no ponto atual,  $x(i)$ , e atualiza-se o valor do argumento usando uma porcentagem do gradiente, ou seja

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

- De tal forma, que a cada **iteração** se tenha
$$f(x(i + 1)) > f(x(i)), i \geq 0.$$
- As iterações se repetem até que o **ponto de máximo** seja atingido, ou seja,  $\nabla f(x(i)) = 0$  e, conseqüentemente, o argumento  $x$  **não sofra mais atualizações** (i.e., o algoritmo convergiu).
  - Chamamos de **convergência** quando a variação da função  $f(x)$  entre iterações consecutivas é muito pequena, ou seja, o valor é praticamente constante.

Mas lembrando o problema da regressão linear, nós não queremos *encontrar o ponto de mínimo da função de erro* ao invés do seu máximo?

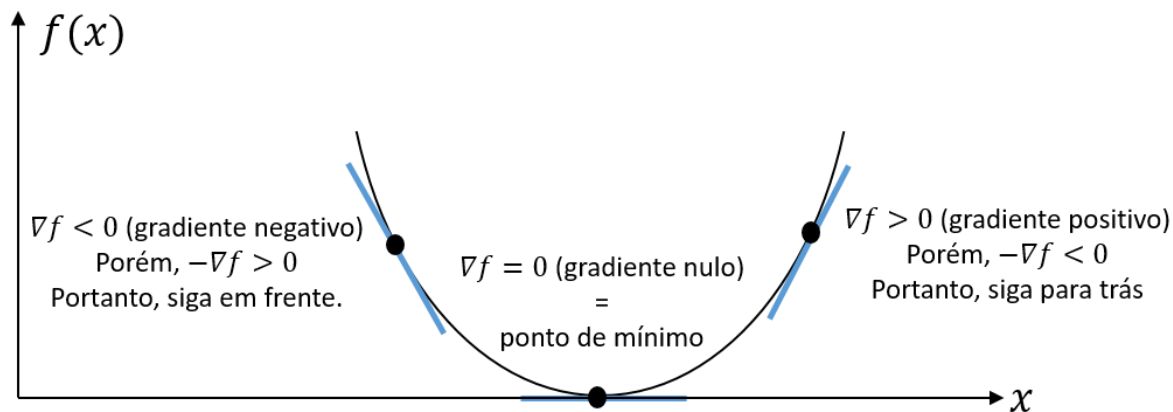
# Algoritmo do gradiente descendente



- Se para encontrarmos o **ponto de máximo** de uma função basta seguirmos a direção apontada pelo vetor gradiente.
- Portanto, para encontrarmos o **ponto de mínimo**, basta seguirmos na **direção aposta à apontada pelo gradiente** em um determinado ponto

$$-\nabla f(x(i)).$$

# Algoritmo do gradiente descendente



- Quando seguimos na ***direção contrária a da máxima taxa de crescimento***, dada pelo ***vetor gradiente***, estamos indo na ***direção de decrescimento mais rápido da função***.
- Assim, um ***algoritmo iterativo de otimização*** que siga na ***direção contrária*** a indicada pelo ***vetor gradiente*** para encontrar o ***ponto de mínimo*** de uma função é chamado de ***gradiente descendente (GD)***.

# Algoritmo do gradiente descendente

- Inicializa-se o argumento  $x(0)$  com um valor arbitrário, em geral, um valor **aleatório**.
- A cada **iteração**,  $i$ , calcula-se o **vetor gradiente** da função  $f(x)$  no ponto atual,  $x(i)$ , e atualiza-se o valor do argumento usando uma porcentagem do gradiente, ou seja

$$x(i + 1) = x(i) \ominus \alpha \nabla f(x(i)), i \geq 0.$$

Única diferença para o gradiente ascendente.

- De tal forma, que a cada **iteração** se tenha
$$f(x(i + 1)) < f(x(i)), i \geq 0.$$
- As iterações se repetem até que o **ponto de mínimo** seja atingido, ou seja,  $\nabla f(x(i)) = 0$  e, conseqüentemente, o argumento  $x$  **não sofra mais atualizações** (i.e., o algoritmo convergiu).
  - **Convergência**: o valor da função  $f(x)$  entre iterações consecutivas é praticamente constante.



# Observação

- Os conceitos vistos até agora foram apresentados para uma função com um único argumento,  $f(x)$ .
- Porém todos eles são válidos para funções com vários argumentos,  $f(x_0, x_1, \dots, x_K)$ .
- Esse será o caso das funções que iremos utilizar em breve.

Como neste curso queremos *minimizar a função de erro*, iremos focar no *gradiente descendente*.

# Características do gradiente descendente

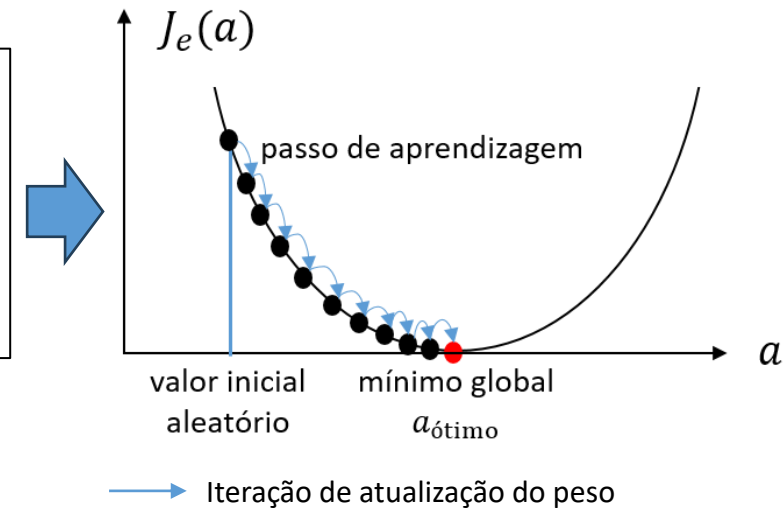
- Algoritmo **genérico de otimização**: pode ser aplicado não apenas a problemas de aprendizado de máquina, mas a **qualquer problema que envolva encontrar os parâmetros que minimizam uma função**.
- O único requisito é que essa função seja **diferenciável**.
  - No caso da regressão linear, a função de erro deve ser diferenciável.
- Escalona melhor do que o método da **equação normal** para grandes conjuntos de dados.
- É de fácil implementação.
- Não é necessário nos preocupar com matrizes mal condicionada, ou seja, matrizes com determinante próximo de 0 (i.e., quase **singulares**).
- Pode ser usado com modelos não-lineares.

*Como aplicamos o algoritmo do **gradiente descendente** ao problema da **regressão linear**?*

*Ou seja, como encontramos os **pesos da função hipótese** usando o gradiente descendente?*

# O algoritmo do gradiente do descendente

$\mathbf{a} \leftarrow$  inicializa o vetor de pesos em um ponto aleatório do espaço de pesos  
**loop** até convergir ou atingir o número máximo de iterações **do**  
     $\nabla J_e(\mathbf{a}) \leftarrow$  cálculo do vetor gradiente,  $\frac{\partial J_e(\mathbf{a})}{\partial \mathbf{a}}$   
     $\mathbf{a} \leftarrow \mathbf{a} - \alpha \times \nabla J_e(\mathbf{a})$  (*regra de atualização dos pesos*)



- **OBS.:** O *passo de aprendizagem*,  $\alpha$ , dita o tamanho do deslocamento dado na direção oposta a apontada pelo **vetor gradiente**.
- Na sequência, encontraremos o **vetor gradiente** da **função de erro**.

# Calculando o vetor gradiente

- Para calcularmos o vetor gradiente, vamos considerar o **EQM** como **função de erro** e a **função do hiperplano** como **função hipótese**:

$$\hat{y}(n) = h(n) = a_0 + a_1 x_1(n) + \cdots + a_K x_K(n) = \sum_{i=0}^K a_i x_i(n) = \mathbf{a}^T \mathbf{x}(n),$$

onde  $K$  é o **número atributos**,  $a_i, \forall i$  e  $x_i, \forall i$  são os pesos e entradas da função hipótese, respectivamente,  $x_0 = 1$  (i.e., atributo de bias) e  $\mathbf{a}$  e  $\mathbf{x}(n)$  são vetores coluna com todos os pesos e entradas, respectivamente.

- Agora podemos encontrar o vetor gradiente.

# Calculando o vetor gradiente

- O **vetor gradiente** da **função de erro em relação aos pesos** é dado por

$$\begin{aligned}\frac{\partial J_e(\mathbf{a})}{\partial \mathbf{a}} &= \frac{\partial}{\partial \mathbf{a}} \left[ \frac{1}{N} \sum_{n=0}^{N-1} (y(n) - \hat{y}(n))^2 \right] \\ &= -\frac{2}{N} \sum_{n=0}^{N-1} [y(n) - \hat{y}(n)] \mathbf{x}(n) = \boxed{-\frac{2}{N} \mathbf{X}^T (\mathbf{y} - \hat{\mathbf{y}})},\end{aligned}$$

Versão matricial do vetor gradiente

onde  $\mathbf{X}$  é a matriz de atributos com dimensão  $(N \times K + 1)$  e  $\mathbf{y}$  e  $\hat{\mathbf{y}}$  são vetores coluna  $(N \times 1)$  com todos os valores esperados e de saída da função hipótese para os  $N$  exemplos coletados, respectivamente.

- Esse cálculo pode ser diretamente estendido a polinômios.

# Atualizando os pesos

- Substituindo o ***vetor gradiente*** na ***equação de atualização dos pesos***, temos

$$\mathbf{a} = \mathbf{a} - \alpha \frac{\partial J_e(\mathbf{a})}{\partial \mathbf{a}} = \mathbf{a} + \alpha \frac{2}{N} \sum_{n=0}^{N-1} [y(n) - \hat{y}(n)] \mathbf{x}(n).$$

- **OBS.:** Por ser constante, o termo  $2/N$  pode ser absorvido por  $\alpha$ .
- Lembrem-se que ***a cada iteração*** do gradiente descendente, precisamos ***calcular o vetor gradiente***.
- Isso envolve ***calcular o somatório da diferença*** entre o rótulo e a saída da função hipótese vezes os atributos ***para todos os  $N$  exemplos do conjunto de treinamento***.
- Assim, ***se o conjunto de treinamento e o modelo forem muito grandes***, o ***treinamento pode ser muito longo e consumir muita CPU e memória***.



# Versões do gradiente descendente

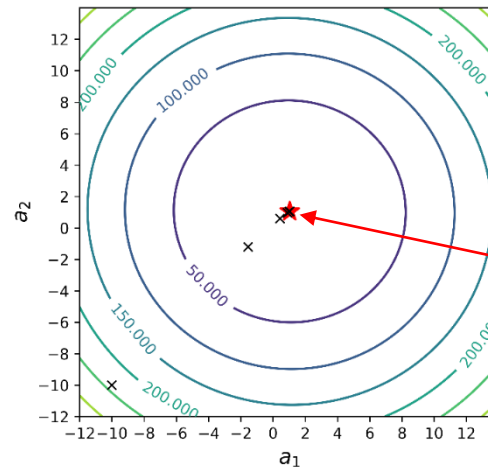
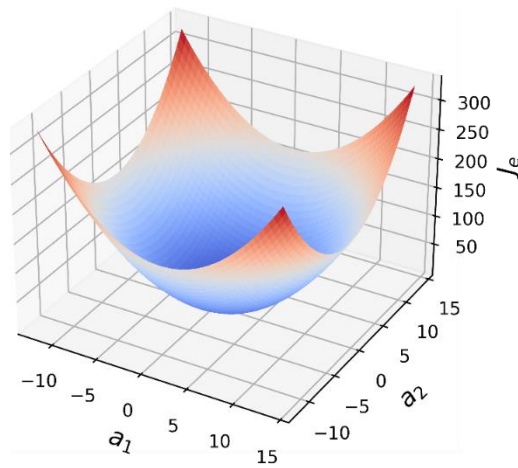
- Portanto, para lidar com essa situação, podemos ter **3 versões diferentes, dependendo da quantidade de exemplos** considerados no somatório do vetor gradiente:
  - ***Gradiente descendente em batelada (GDB)***
  - ***Gradiente descendente estocástico (GDE)***
  - ***Gradiente descendente em mini-lotes (GDML)***

# Gradiente descendente em batelada

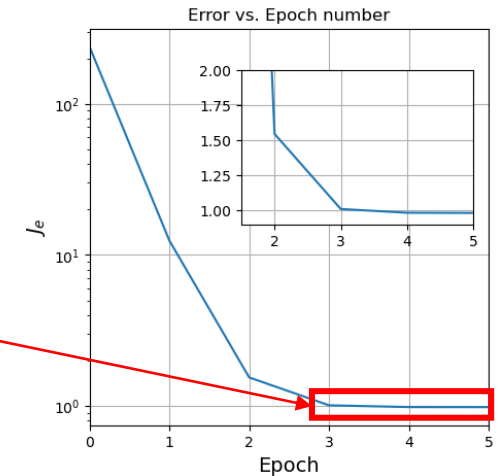
$$\mathbf{a} = \mathbf{a} + \alpha \frac{2}{N} \sum_{n=0}^{N-1} [y(n) - \hat{y}(n)] \mathbf{x}(n).$$

- Considera **todos os exemplos** do conjunto de treinamento para calcular o vetor gradiente.
- Pode ser **computacionalmente custoso** dependendo do tamanho do modelo e do conjunto de dados.
  - Por processar todos os exemplos, pode ser lento e consumir muita CPU e memória com conjuntos muito grandes.
- **Convergência** para o mínimo global é **garantida** quando a função de erro é **convexa**.
- É a versão que **obtém os melhores resultados**.

# Características do GD em batelada



convergência



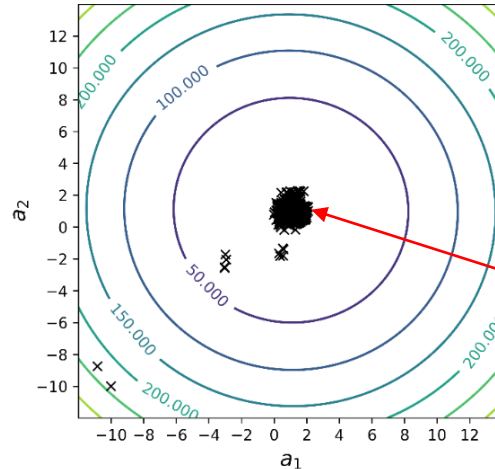
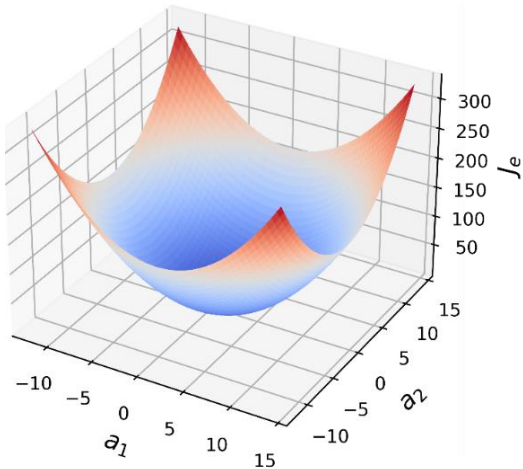
- Por usar todos os exemplos, **segue diretamente**, sem alterar a direção, para o **ponto de mínimo**.
  - Nesse exemplo, segue uma linha reta entre  $a_1$  e  $a_2$  pois a taxa de decrescimento da superfície de erro é igual para os dois pesos (contornos são circulares).
- **Convergência é garantida** dado que o **passo de aprendizagem não seja muito grande** e se **espere tempo suficiente**.
  - Não fica “oscilando” em torno do ponto de mínimo após alcançá-lo, pois o vetor gradiente neste ponto é praticamente nulo.

# Gradiente descendente estocástico

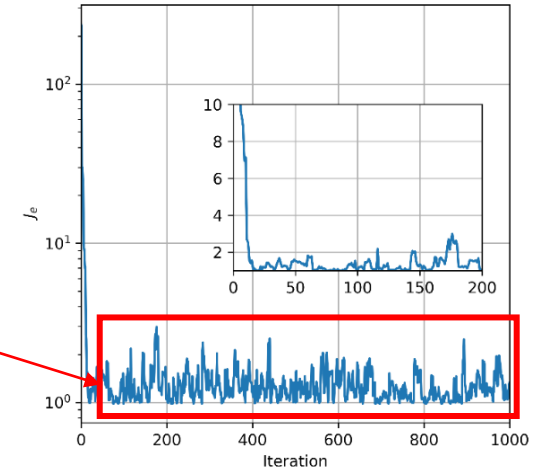
$$\mathbf{a} = \mathbf{a} + \alpha 2[y(n_{\text{random}}) - \hat{y}(n_{\text{random}})]\mathbf{x}(n_{\text{random}}).$$

- Utiliza a cada iteração de atualização **apenas um exemplo** do conjunto de treinamento para calcular uma **estimativa estocástica do gradiente**.
  - É estocástica pois a cada iteração toma-se **uma amostra aleatória** do conjunto de treinamento para calcular a estimativa do vetor gradiente.
- Por usar uma estimativa, **não segue diretamente a direção de máxima declividade da função de erro, mudando de direção várias vezes**.
- Quando os **dados de treinamento estão contaminados com ruído**, a **estimativa** do gradiente é **ruidosa**, fazendo com que a **convergência não ocorra** ou **não seja garantida**.
  - O algoritmo pode **oscilar** em torno do ponto de mínimo **sem nunca convergir**.
- Entretanto, é mais rápido, usando menos CPU e memória do que o GDB.

# Características do GD estocástico



Não  
converge



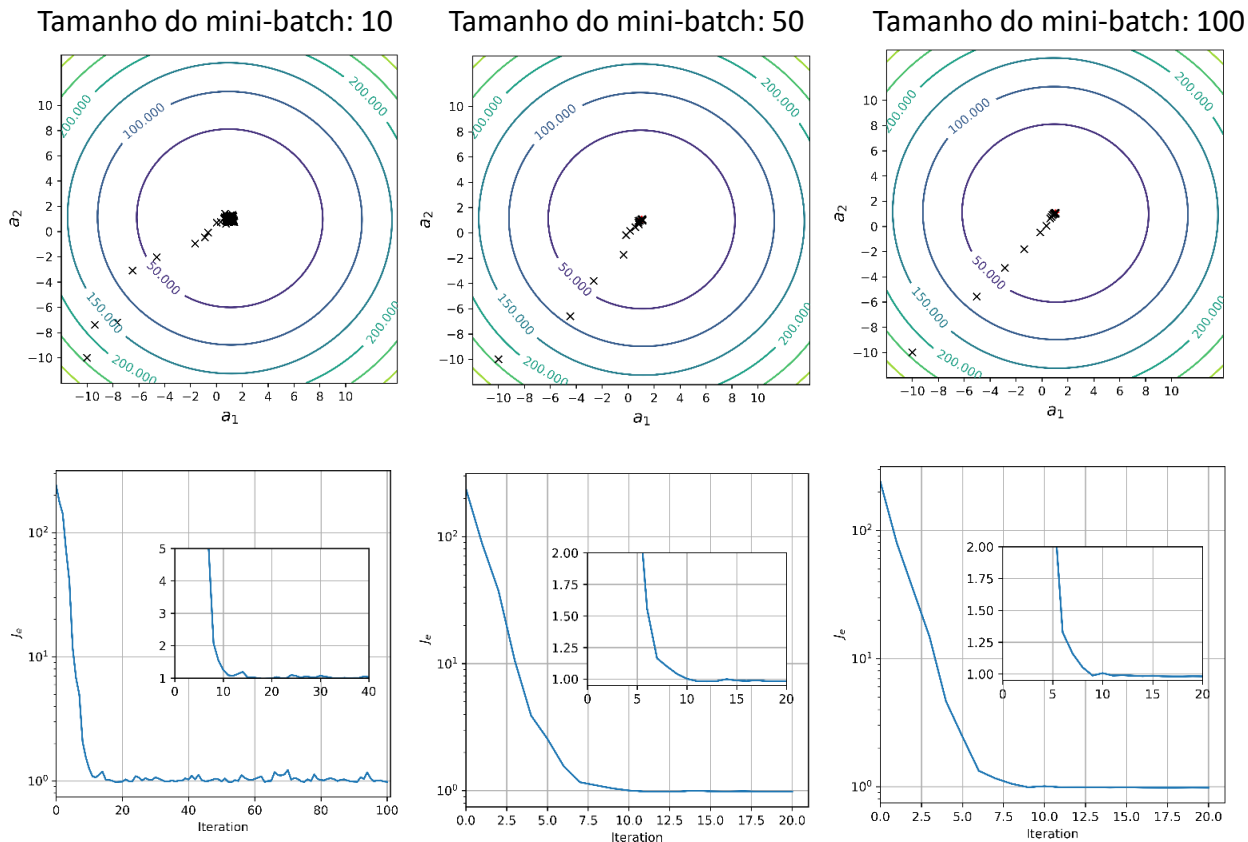
- ***Apresenta um caminho *irregular* para o mínimo, mudando de direção várias vezes.***
- ***Quando as *amostras contém ruído (caso acima), não converge para o ponto de mínimo, “oscila” em torno dele.****
- Essa oscilação também pode ser vista na curva de erro.
- Algumas técnicas podem ser usadas para torná-lo mais comportado e talvez convergir: redução do passo, *early-stop*, momentum, etc.

# Gradiente descendente em mini-lotes

$$\mathbf{a} = \mathbf{a} + \alpha \frac{2}{MB} \sum_{n=0}^{MB-1} [y(n) - \hat{y}(n)] \mathbf{x}(n).$$

- Utiliza a cada iteração um **subconjunto aleatório de exemplos**, de tamanho  $MB$ , do conjunto de treinamento para o cálculo do gradiente.
- Em geral,  $1 < MB < N$ , portanto é mais rápido que o GDB e mais preciso e estável do que o GDE.
- Porém, por  $MB$  ser variável, essa versão é vista como uma **generalização** das duas versões anteriores, pois  $MB$  pode ser feito igual a 1 ou  $N$ .
- Em caso de **amostras ruidosas**, a convergência depende do tamanho de  $MB$ , **quanto maior, melhor é a estimativa** do vetor gradiente e, consequentemente, **maior a chance de convergência**.

# Características do GD em mini-lotes



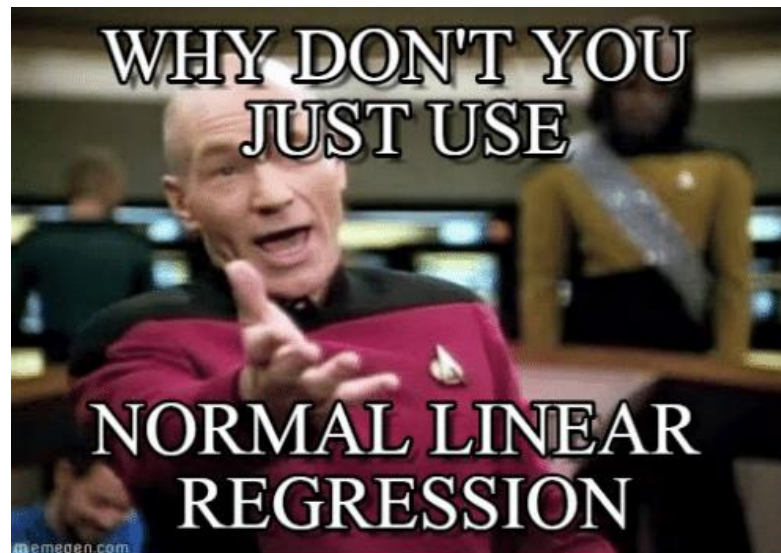
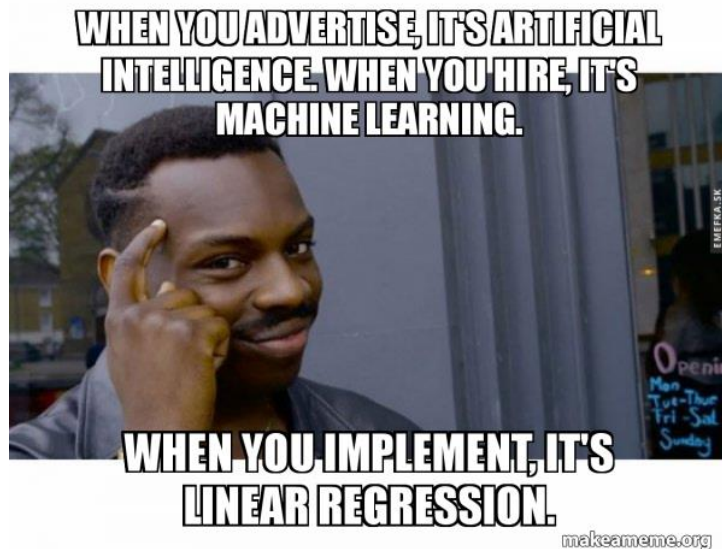
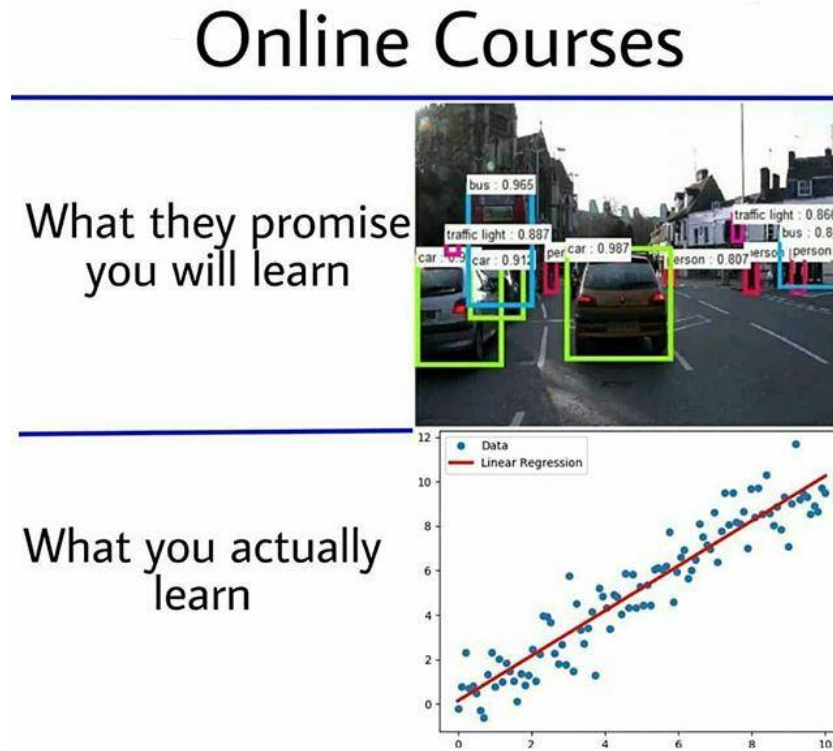
- Vejam que conforme  $MB$  aumenta,
  - o progresso se torna menos irregular do o do GDE,
  - e a oscilação ao redor do ponto de mínimo diminui.
- Tem comportamento mais próximo do GD em batelada para mini-lotes maiores.
- Para  $MB$  pequenos, pode se beneficiar de técnicas para torná-lo mais comportado e talvez convergir.
  - Essas técnicas podem ajudar a **balancear rapidez e convergência**.

# 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 -> Recordings -> Laboratório #3
  - Se atentem aos prazos de entrega.
  - [Instruções para resolução e entrega dos laboratórios.](#)



Obrigado!



# Anexo I:

## Cálculo do vetor gradiente

# Cálculo do vetor gradiente

Considerando o hiperplano como a função hipótese

$$\hat{y}(n) = \mathbf{a}^T \mathbf{x}(n).$$

O vetor gradiente é calculado como

$$\frac{\partial J_e(\mathbf{a})}{\partial \mathbf{a}} = \left[ \frac{\partial J_e(\mathbf{a})}{\partial a_0} \quad \dots \quad \frac{\partial J_e(\mathbf{a})}{\partial a_K} \right]^T.$$

Assim, o vetor gradiente da função de erro em relação aos pesos é dado por

$$\frac{\partial J_e(\mathbf{a})}{\partial \mathbf{a}} = \frac{\partial}{\partial \mathbf{a}} \left[ \frac{1}{N} \sum_{n=0}^{N-1} (y(n) - \hat{y}(n))^2 \right].$$

# Cálculo do vetor gradiente

Como a operação de derivada é distributiva, podemos reescrever a equação acima como

$$\frac{\partial J_e(\mathbf{a})}{\partial \mathbf{a}} = \frac{1}{N} \sum_{n=0}^{N-1} \frac{\partial (y(n) - \hat{y}(n))^2}{\partial \mathbf{a}}.$$

Substituindo a função hipótese na equação acima, temos

$$\frac{\partial J_e(\mathbf{a})}{\partial \mathbf{a}} = \frac{1}{N} \sum_{n=0}^{N-1} \frac{\partial (y(n) - \mathbf{a}^T \mathbf{x}(n))^2}{\partial \mathbf{a}}.$$

# Cálculo do vetor gradiente

Aplicando a regra da cadeia, reescrevemos a equação anterior como

$$\frac{\partial J_e(\mathbf{a})}{\partial \mathbf{a}} = -\frac{2}{N} \sum_{n=0}^{N-1} (y(n) - \mathbf{a}^T \mathbf{x}(n)) \frac{\partial \mathbf{a}^T \mathbf{x}(n)}{\partial \mathbf{a}}.$$

Sabendo que a derivada de  $\frac{\partial \mathbf{a}^T \mathbf{x}(n)}{\partial \mathbf{a}}$  é igual a  $\mathbf{x}(n)$ , reescrevemos a equação anterior como

$$\frac{\partial J_e(\mathbf{a})}{\partial \mathbf{a}} = -\frac{2}{N} \sum_{n=0}^{N-1} [y(n) - \hat{y}(n)] \mathbf{x}(n).$$

# Cálculo do vetor gradiente

Fazendo  $y(n) - \hat{y}(n) = d(n)$

$$\begin{aligned}\frac{\partial J_e(\mathbf{a})}{\partial \mathbf{a}} &= -\frac{2}{N} \sum_{n=0}^{N-1} [y(n) - \hat{y}(n)] \mathbf{x}(n) \\ &= -\frac{2}{N} \left\{ d(0) \begin{bmatrix} x_0(0) \\ \vdots \\ x_K(0) \end{bmatrix} + d(1) \begin{bmatrix} x_0(1) \\ \vdots \\ x_K(1) \end{bmatrix} + \cdots + d(N-1) \begin{bmatrix} x_0(N-1) \\ \vdots \\ x_K(N-1) \end{bmatrix} \right\} \\ &= -\frac{2}{N} \begin{bmatrix} d(0)x_0(0) + d(1)x_0(1) + \cdots + d(N-1)x_0(N-1) \\ \vdots \\ d(0)x_K(0) + d(1)x_K(1) + \cdots + d(N-1)x_K(N-1) \end{bmatrix}.\end{aligned}$$

Notem que a equação acima é um **vetor coluna** com dimensão  $K + 1 \times 1$ .

# Cálculo do vetor gradiente

Podemos reescrever a equação (i.e., vetor) anterior como uma multiplicação matricial

$$\frac{\partial J_e(\mathbf{a})}{\partial \mathbf{a}} = -\frac{2}{N} \begin{bmatrix} x_0(0) & x_0(1) & \cdots & x_0(N-1) \\ \vdots & \vdots & & \vdots \\ x_K(0) & x_K(1) & \cdots & x_K(N-1) \end{bmatrix} \begin{bmatrix} d(0) \\ d(1) \\ \vdots \\ d(N-1) \end{bmatrix}$$

Percebam que temos a multiplicação de uma matriz com dimensão  $K + 1 \times N$  por um vetor coluna de dimensão  $N \times 1$ .

A matriz contém em cada linha todos os valores de  $n = 0$  a  $n = N - 1$  de um **único** atributo.

O vetor contém em cada linha a diferença  $d(n) = y(n) - \hat{y}(n)$  para  $n = 0$  até  $n = N - 1$ .



# Cálculo do vetor gradiente

Se definirmos uma matriz que contém todos os  $N$  exemplos de todos os  $K + 1$  atributos e que tem dimensão  $N \times K + 1$

$$\mathbf{X} = \begin{bmatrix} x_0(0) & \cdots & x_K(0) \\ x_0(1) & \cdots & x_K(1) \\ \vdots & & \vdots \\ x_0(N-1) & \cdots & x_K(N-1) \end{bmatrix},$$

e dois vetores coluna com dimensões  $N \times 1$  contendo todos os valores esperados (i.e., rótulos) e todos os valores preditos

$$\mathbf{y} = \begin{bmatrix} y(0) \\ \vdots \\ y(N-1) \end{bmatrix} \quad \text{e} \quad \hat{\mathbf{y}} = \begin{bmatrix} \hat{y}(0) \\ \vdots \\ \hat{y}(N-1) \end{bmatrix}$$

# Cálculo do vetor gradiente

Usando a matriz e os vetores definidos no slide anterior, podemos reescrever o vetor gradiente como

$$\begin{aligned} & \frac{\partial J_e(\mathbf{a})}{\partial \mathbf{a}} \\ &= -\frac{2}{N} \begin{bmatrix} x_0(0) & x_0(1) & \cdots & x_0(N-1) \\ \vdots & \vdots & & \vdots \\ x_K(0) & x_K(1) & \cdots & x_K(N-1) \end{bmatrix} \left( \begin{bmatrix} y(0) \\ y(1) \\ \vdots \\ y(N-1) \end{bmatrix} - \begin{bmatrix} \hat{y}(0) \\ \hat{y}(1) \\ \vdots \\ \hat{y}(N-1) \end{bmatrix} \right) \\ &= -\frac{2}{N} \mathbf{X}^T (\mathbf{y} - \hat{\mathbf{y}}) \end{aligned}$$

O resultado da multiplicação matricial acima continua resultando em um **vetor coluna** com dimensão  $K + 1 \times 1$ , ou seja,  $(K + 1 \times N) \times (N \times 1) = K + 1 \times 1$ .

# Equação de atualização dos pesos

Utilizando o resultado anterior, podemos reescrever a equação de atualização dos pesos como

$$\begin{aligned}\mathbf{a} &= \mathbf{a} - \alpha \frac{\partial J_e(\mathbf{a})}{\partial \mathbf{a}} \\ &= \mathbf{a} + \alpha \frac{2}{N} \mathbf{X}^T (\mathbf{y} - \hat{\mathbf{y}}).\end{aligned}$$

A soma acima deve resultar em um vetor coluna com dimensão  $K + 1 \times 1$ , pois esta é a dimensão dos dois vetores sendo somados.

Lembrem-se que  $K + 1 \times 1$  é a dimensão do vetor  $\mathbf{a}$ , o qual contém todos os pesos do modelo e que  $\frac{2}{N} \mathbf{X}^T (\mathbf{y} - \hat{\mathbf{y}})$  tem dimensão  $K + 1 \times 1$  também.

# Equação de atualização dos pesos

Podemos reescrever a equação de atualização dos pesos como

$$\begin{aligned}\mathbf{a} &= \mathbf{a} - \alpha \frac{\partial J_e(\mathbf{a})}{\partial \mathbf{a}} = \begin{bmatrix} a_0 \\ \vdots \\ a_K \end{bmatrix} - \alpha \begin{bmatrix} \frac{\partial J_e(\mathbf{a})}{\partial a_0} \\ \vdots \\ \frac{\partial J_e(\mathbf{a})}{\partial a_K} \end{bmatrix} \\ &= \begin{bmatrix} a_0 \\ \vdots \\ a_K \end{bmatrix} + \alpha \frac{2}{N} \begin{bmatrix} x_0(0) & x_0(1) & \cdots & x_0(N-1) \\ \vdots & \vdots & & \vdots \\ x_K(0) & x_K(1) & \cdots & x_K(N-1) \end{bmatrix} \left( \begin{bmatrix} y(0) \\ y(1) \\ \vdots \\ y(N-1) \end{bmatrix} - \begin{bmatrix} \hat{y}(0) \\ \hat{y}(1) \\ \vdots \\ \hat{y}(N-1) \end{bmatrix} \right) \\ &= \begin{bmatrix} a_0 \\ \vdots \\ a_K \end{bmatrix} + \alpha \frac{2}{N} \left\{ \begin{bmatrix} d(0)x_0(0) + d(1)x_0(1) + \cdots + d(N-1)x_0(N-1) \\ \vdots \\ d(0)x_K(0) + d(1)x_K(1) + \cdots + d(N-1)x_K(N-1) \end{bmatrix} \right\}.\end{aligned}$$

# Anexo II:

## Cálculo do vetor gradiente de uma função hipótese com 2 pesos

# Cálculo do 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

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

# Cálculo do vetor gradiente

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.

FIGURAS



