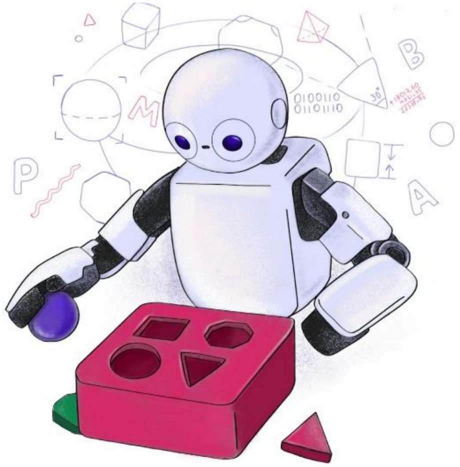
TP557 - Tópicos avançados em IoT e Machine Learning: *Minimizando o erro*



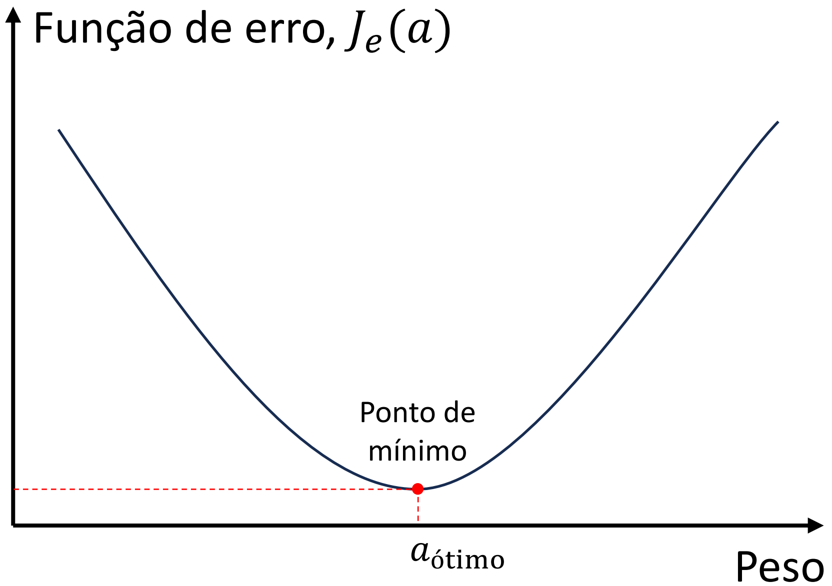
Inatel

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

O que vamos ver?

- Anteriormente, vimos como funciona o processo (*loop*) de treinamento.
    1. Damos um palpite.
    2. Medimos a precisão desse palpite com a função de erro.
    3. Então usamos a informação do erro para dar outro palpite, esperando que ele seja um pouco melhor do que o anterior.
- Em geral, esse processo se repete até que o erro seja minimizado.
- A ideia é que quanto menor o erro, mais preciso é o seu palpite.
- Portanto, neste tópico, exploraremos como minimizar o erro.

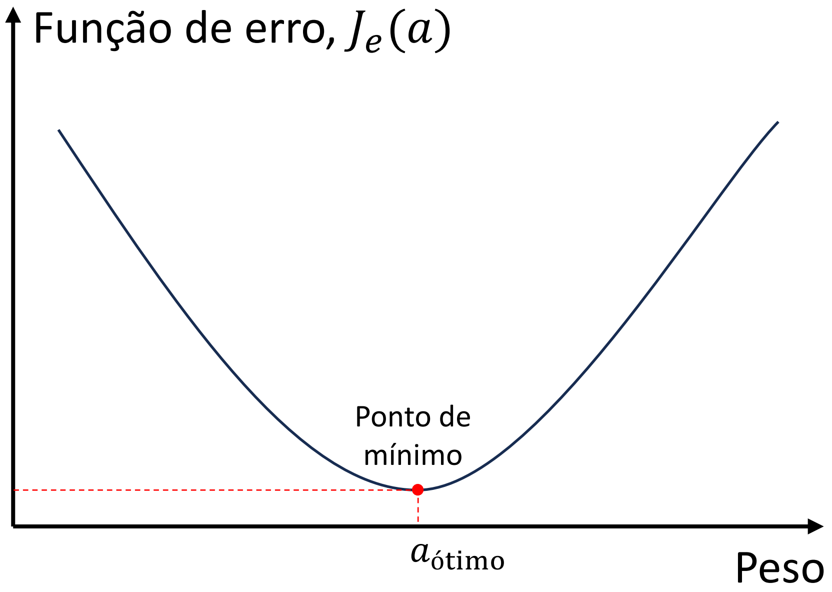
Vetores gradiente



A função de erro, $J_e(a)$, quantifica a diferença entre y e $\hat{y}$.

- Vamos primeiro ver com vetores gradiente nos ajudam a minimizar o erro.
- Ou seja, entender como eles nos ajudam a encontrar o ***ponto de mínimo da função de erro***.
- Consequentemente, entenderemos como o algoritmo de otimização (ou treinamento) dos modelos funciona.
- **OBS.**: Vamos usar $J_e(a)$ para definir uma função de erro genérica.

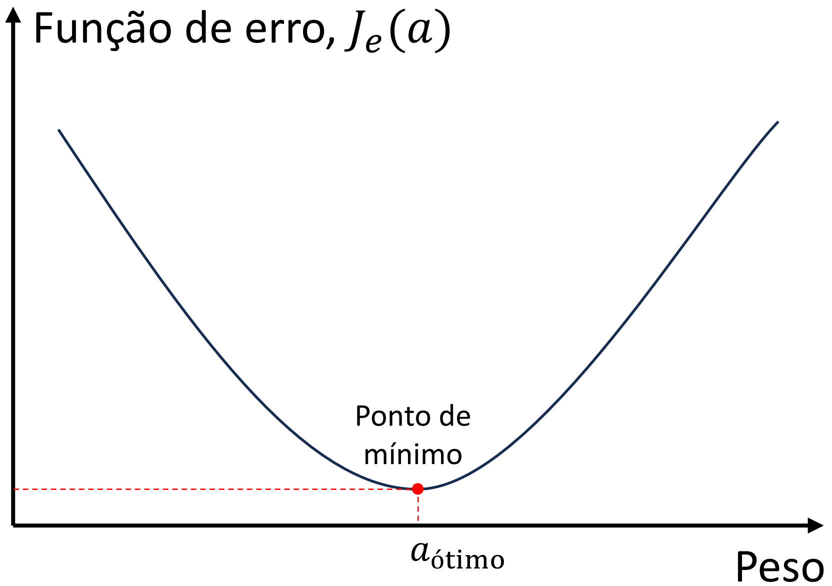
Função de erro



$$J_e(a) = \text{MSE} = \frac{1}{N}\sum_{n=1}^{N}(\hat{y}_n(a) - y_n)^2$$

- Lembrem-se que a função de erro é função dos pesos.
- Ou seja, o erro varia se variarmos os valores dos pesos.
- Portanto, variando os pesos, conseguimos, em alguns casos, visualizar a superfície de erro.
- O ponto mais baixo dessa superfície nos dá os valores dos pesos que minimizam a função de erro.

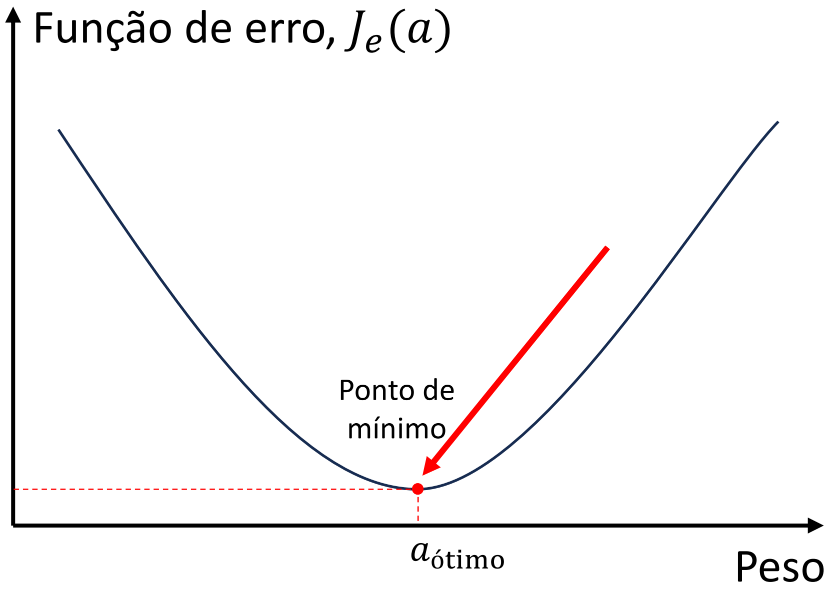
Formato da função de erro



$$J_e(a) = \text{MSE} = \frac{1}{N}\sum_{n=1}^{N}(\hat{y}_n(a) - y_n)^2$$

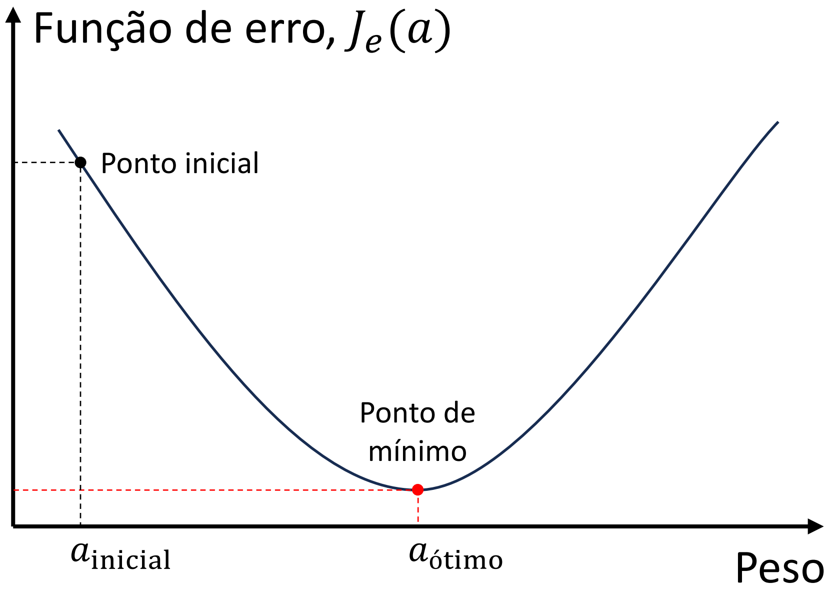
- Lembrem-se que a função de erro que usamos anteriormente, a ***função do EQM, é quadrática***.
- E como vimos no exemplo, ***funções quadráticas*** têm a ***forma de parábolas convexas***.
- A ***convexidade*** é ***importante*** pois ***garante*** que a ***função tenha apenas um ponto de mínimo, o mínimo global***.
- Isso permite que encontremos a solução ótima de forma mais eficiente.

Ponto de mínimo de uma função convexa



- Se queremos encontrar o ***mínimo*** da função, basta buscarmos a ***parte mais baixa da parábola***.
- Não importa quais sejam os valores dos pesos e onde a função de erro é plotada no gráfico, nós ***sempre teremos certeza de que o mínimo está na parte inferior da parábola***.

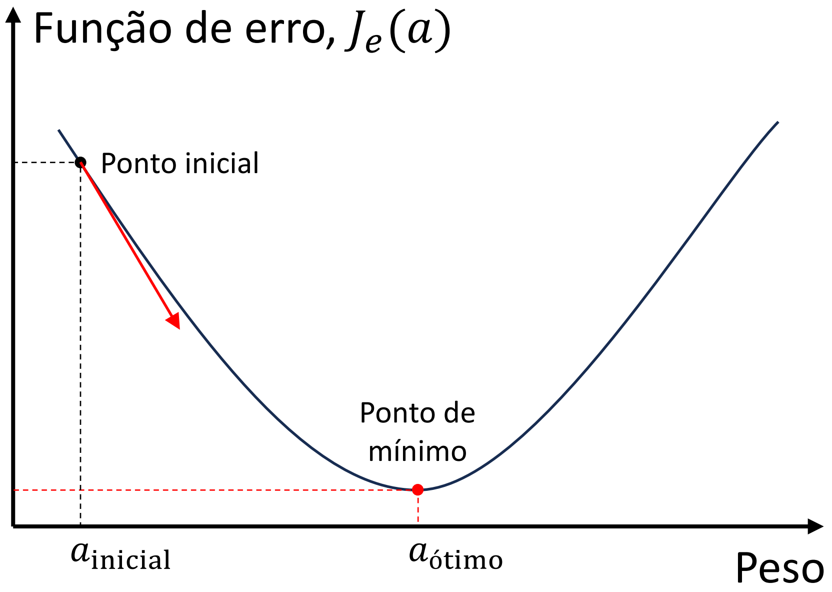
O erro indica o caminho a ser seguido



- Assim, se dermos um palpite (***ponto inicial***) sobre os valores dos ***pesos da função hipótese***, como mostrado ao lado, e calcularmos o ***erro***, ele será ***alto*** e, consequentemente, saberemos que ***estamos longe do ponto de mínimo***.
- Portanto, quanto menor o erro, mais próximo estaremos do ponto ótimo.
- **OBS.**: Em geral, o erro nunca será igual a 0 devido aos dados estarem corrompidos por ruído.

Como encontramos o ponto de mínimo através do erro?

Vetor gradiente

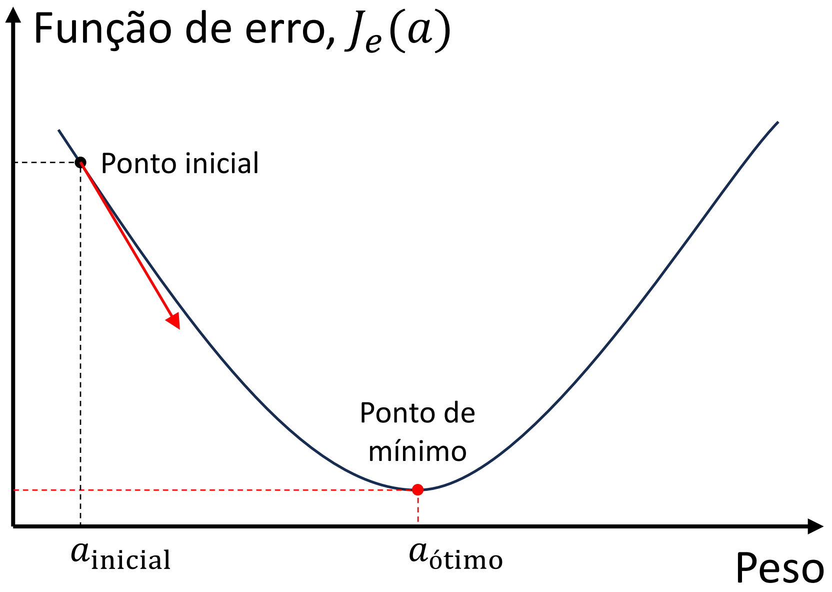


- Se nós diferenciarmos a função de erro em um ponto qualquer em relação aos pesos, nós obtemos o ***vetor gradiente***

$$\nabla J_e(\boldsymbol{a}) = \frac{\partial J_e(\boldsymbol{a})}{\partial \boldsymbol{a}}.$$

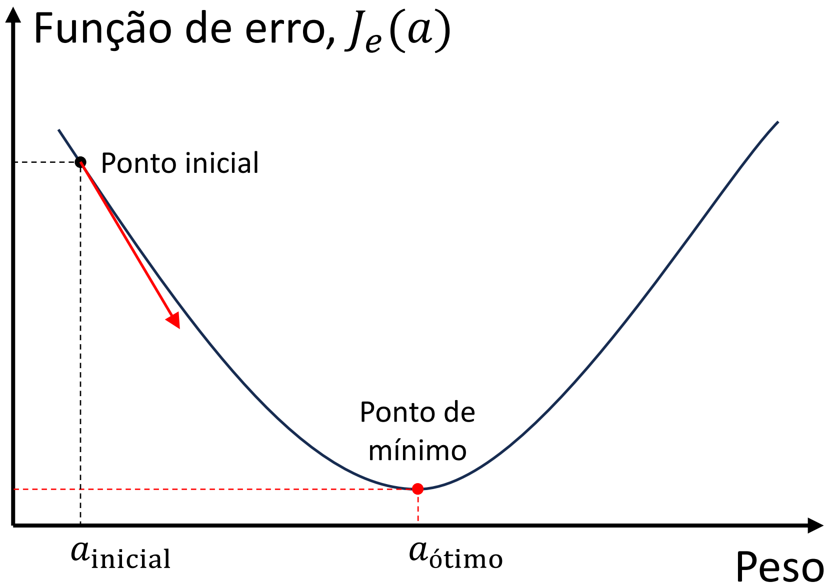
- Ele sempre aponta na ***direção de maior crescimento da função*** a partir de um determinado ponto.

Vetor gradiente



- O gradiente pode ser também interpretado como a ***inclinação*** de uma ***reta tangente à curva*** no ponto onde ele é calculado.
  - Quanto ***maior o valor absoluto*** do gradiente***, mais inclinada é a reta tangente*** naquele ponto.
  - Portanto, ***um valor igual a 0 indica inclinação nula***.
  - Onde isso ocorre? ***Em pontos de máximo e de mínimo***.

Vetor gradiente



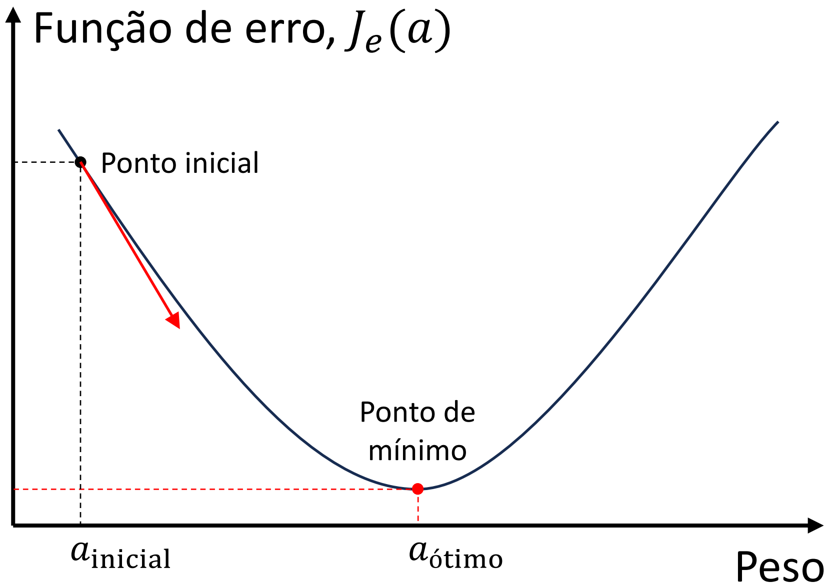
O objetivo é minimizar a função de erro ***indo na direção contrária*** a indicada pelo gradiente.

- Porém, queremos ***encontrar o mínimo*** da função, o que devemos fazer?
- Basta irmos na ***direção oposta*** a indicada pelo vetor gradiente (i.e., negativo do gradiente)

$$-\nabla J_e(\boldsymbol{a}) = -\frac{\partial J_e(\boldsymbol{a})}{\partial \boldsymbol{a}}$$

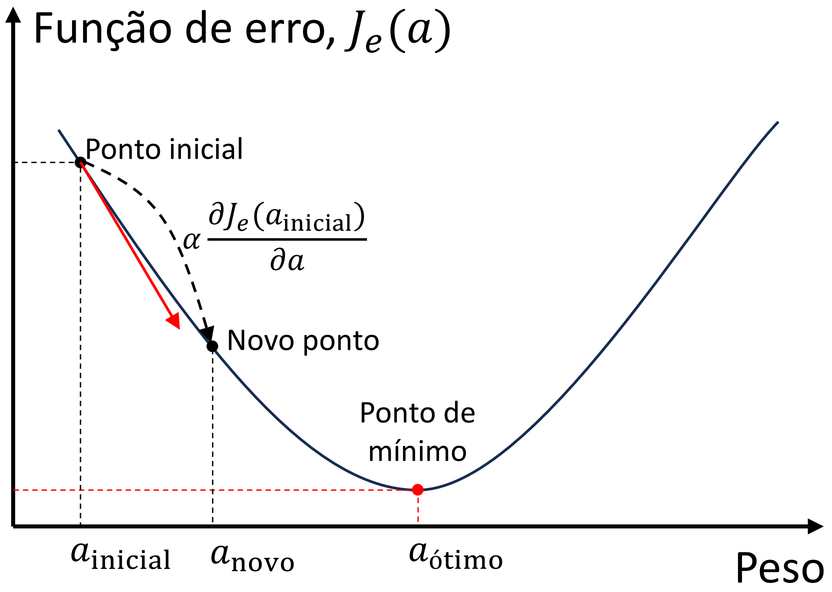
- O ***negativo do gradiente aponta para a direção de maior decréscimo da função*** a partir de um dado ponto.

Qual a distância até o mínimo?



- Entretanto, o gradiente ***não dá informação sobre a distância até o ponto de mínimo***, mas pelo menos sabemos a direção correta.
- Podemos fazer a analogia com uma bola solta em uma ladeira.
- A gravidade dá a direção até a parte mais baixa da ladeira, mas não dá a distância até lá.

Passo de aprendizagem

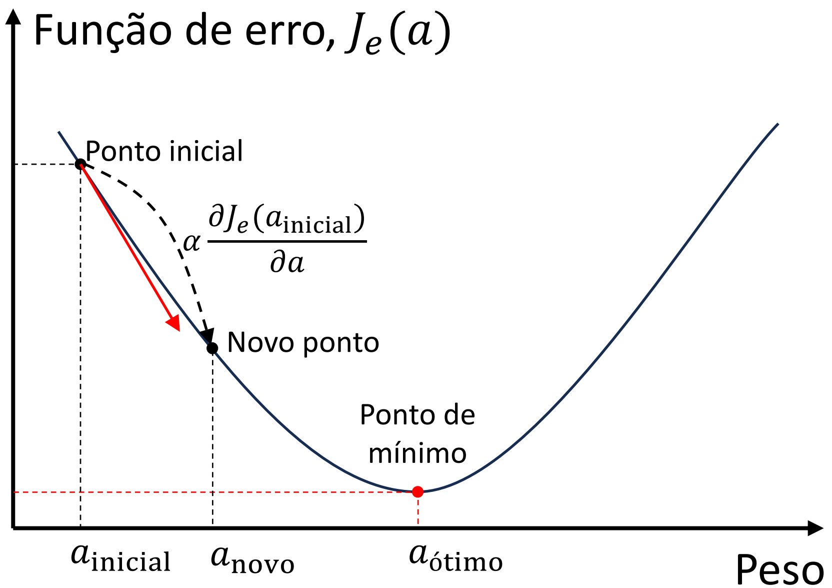


$$a_{\text{novo}} = a_{\text{inicial}} - \alpha \frac{\partial J_e(a_{\text{inicial}})}{\partial a}$$

sentido aposto ao apontado pelo gradiente

- Portanto, se quisermos ***seguir até o ponto de mínimo*** a partir de um ponto qualquer, podemos ***dar um passo na direção apontada pelo gradiente***.
- Nós sabemos a direção do mínimo e podemos escolher o ***tamanho do passo*** para darmos naquela direção.
- O passo de aprendizagem determina a ***porcentagem do gradiente que é adicionada aos pesos***.
- A equação ao lado é chamada de ***equação de atualização dos pesos***.

Passo de aprendizagem

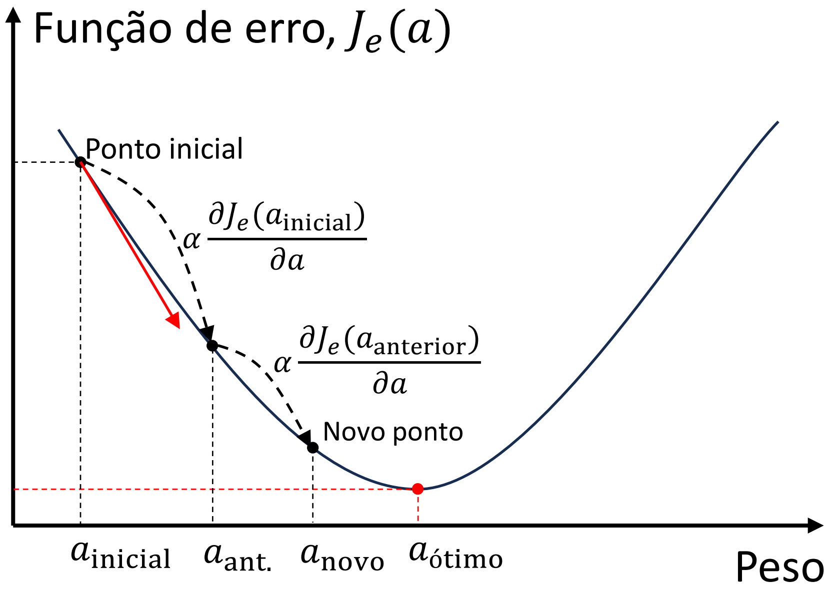


$$a_{\text{novo}} = a_{\text{inicial}} - \underbrace{\alpha \frac{\partial J_e(a_{\text{inicial}})}{\partial a}}_{\text{Termo de atualização}}$$

sentido aposto ao apontado pelo gradiente

- O ***tamanho do passo*** é frequentemente chamado de ***taxa ou passo de aprendizagem*** e é, normalmente, denotado pela letra grega α.
- Se o peso atual está à esquerda do mínimo, o ***termo de atualização*** faz com que o novo peso seja maior do que o anterior.
- Se o peso atual está à direita do mínimo, o ***termo de atualização*** faz com que o novo peso seja menor do que o anterior.

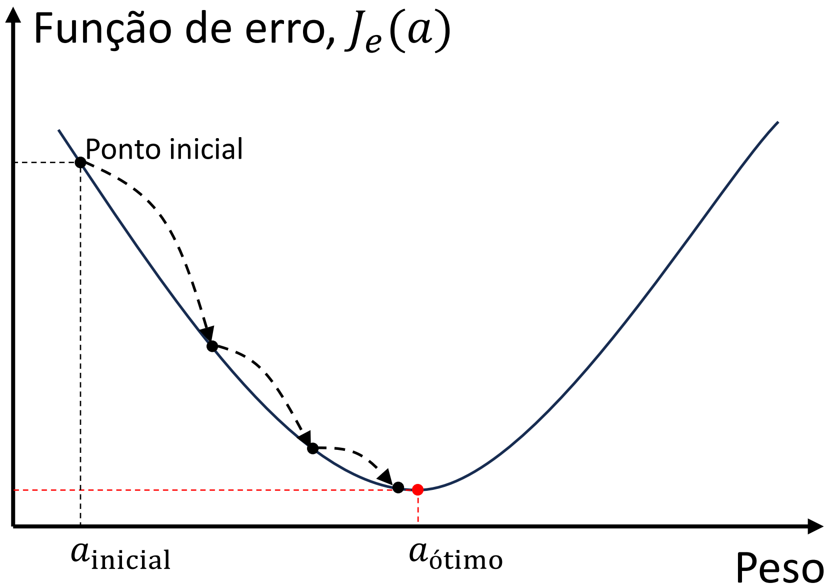
Otimização iterativa



$$\underbrace{a_{\text{novo}} = a_{\text{anterior}} - \alpha \frac{\partial J_e(a_{\text{anterior}})}{\partial a}}_{\text{Equação de atualização dos pesos}}$$

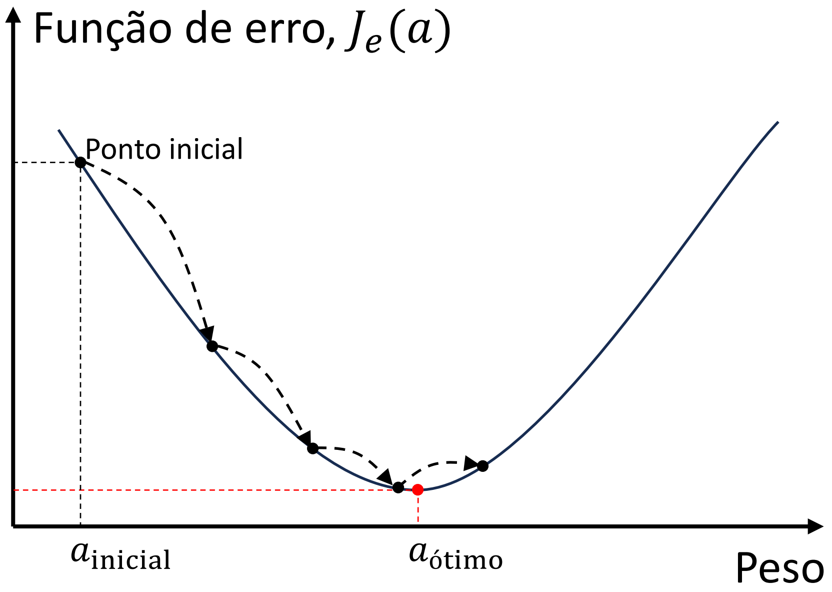
- Portanto, dada a ***direção do gradiente e um passo de aprendizagem***, agora podemos ***iterativamente*** dar passos em direção ao ponto de mínimo.
- A cada ***iteração*** calculamos o gradiente no ponto atual, atualizamos os pesos com uma porcentagem do gradiente e calculamos o gradiente no novo ponto.
- ***Repetimos*** esse processo ***até*** que a ***inclinação da reta tangente ao ponto atual se torne igual a 0***, indicando que o ponto de mínimo foi atingido.
- ***O que ocorre quando o gradiente é 0?***

Tamanho do passo de aprendizagem



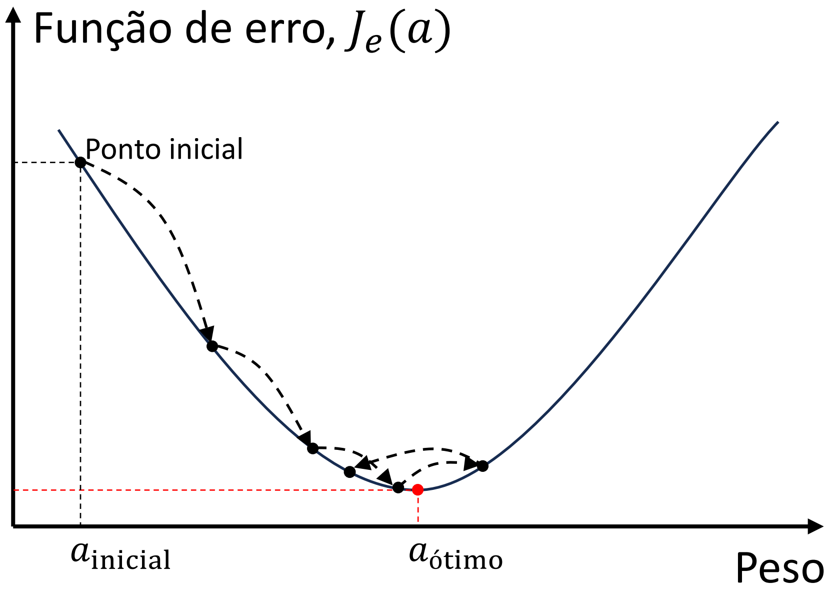
- O objetivo é que a cada nova iteração, nos movamos para mais e mais perto do ponto de mínimo, $a_{\text{ótimo}}$.
- Porém, devemos tomar ***cuidado***, com o ***tamanho do passo de aprendizagem***.
- O valor do passo de aprendizagem é um ***hiperparâmetro*** crucial para o GD.
  - ***Hiperparâmetros*** são parâmetros do modelo que ***não são aprendidos durante o treinamento***, mas sim definidos pelo desenvolvedor antes do treinamento.
- Ele influencia a velocidade e a convergência do treinamento.

Tamanho do passo de aprendizagem



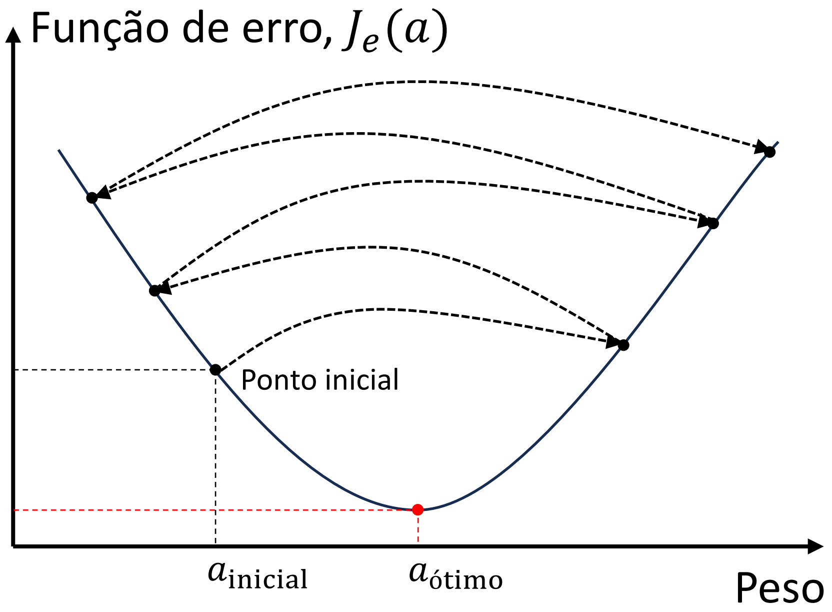
- Se o passo de aprendizagem for ***muito grande***, podemos ultrapassar o ponto de mínimo.

Tamanho do passo de aprendizagem



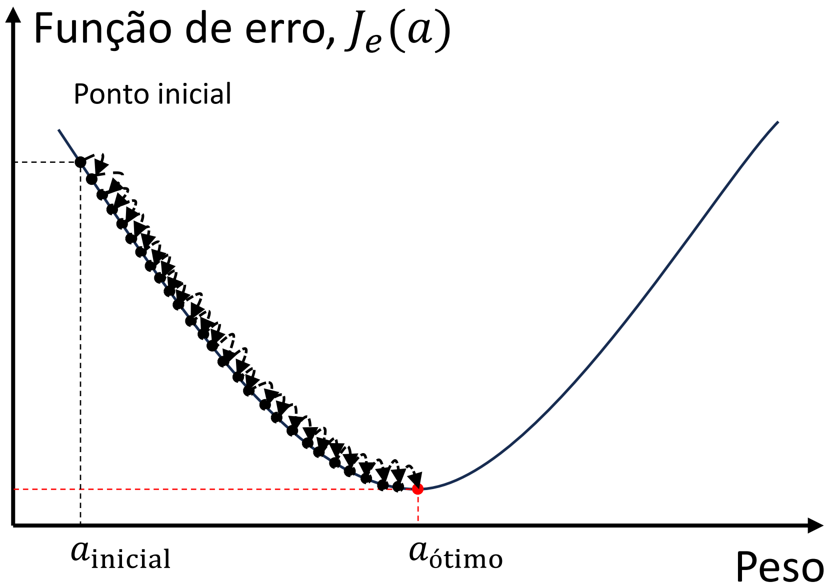
- Se o passo de aprendizagem for ***muito grande***, o processo de otimização pode ficar ***ziguezagueando*** de um lado para o outro do fundo da função ***sem nunca atingir o ponto de mínimo***.

Tamanho do passo de aprendizagem



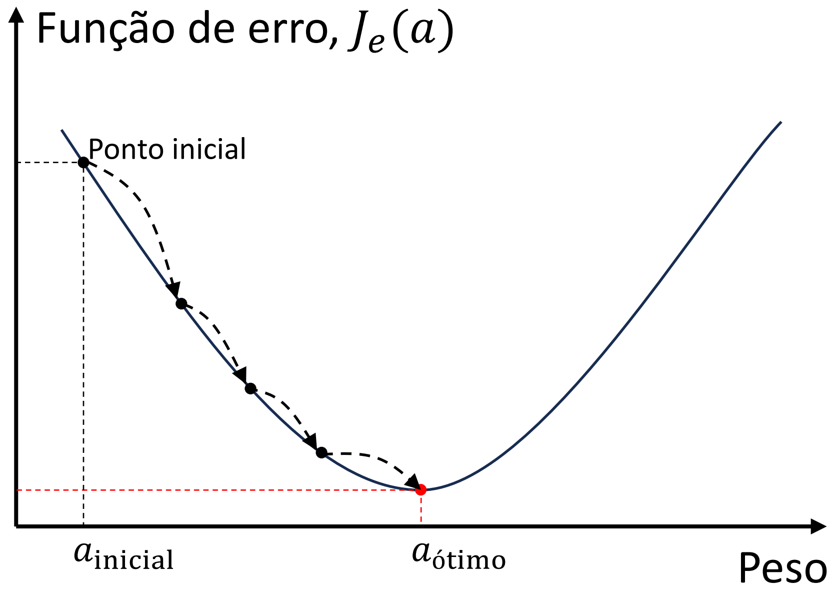
- Dependendo do quão grande for o valor do passo de aprendizagem, pode ocorrer até a ***divergência*** ao invés da convergência.
- Ou seja, ao invés de se aproximar do ponto de mínimo, o algoritmo ***se distancia*** dele a cada iteração.
- Se isso ocorrer, após algumas iterações, ocorre o estouro da precisão numérica das variáveis envolvidas na regra de atualização dos pesos.

Tamanho do passo de aprendizagem



- Outra questão, ***menos problemática*** que passos grandes, é a situação oposta.
- Passos ***muito pequenos*** fazem com que se ***leve muito tempo***, i.e., iterações, para atingir o ponto de mínimo.
- Se cada iteração levar um tempo razoável para ser executada, o tempo necessário para se atingir o ponto de mínimo pode ser muto grande.
- Porém, ***se esperarmos*** tempo suficiente, a ***convergência é garantida***.

Qual o tamanho de passo de aprendizagem usar?



- É importante escolhermos um passo de aprendizagem que ***acelere a convergência sem causar oscilações em torno do ponto de mínimo***.
- Em geral, um bom valor para o passo é encontrado por ***tentativa e erro***.
- Uma forma mais avançada é ajustar o passo ao longo das iterações.
  - Usamos ***valores grandes no início***, em pontos distantes do mínimo, e o ***reduzimos gradualmente*** ao longo das iterações.

Formas de se ajustar o passo de aprendizagem

Taxa constante

$$\boldsymbol{a}(i+1) = \boldsymbol{a}(i) - \alpha \frac{\partial J_e(\boldsymbol{a}(i))}{\partial \boldsymbol{a}}$$

Decaimento linear e exponencial

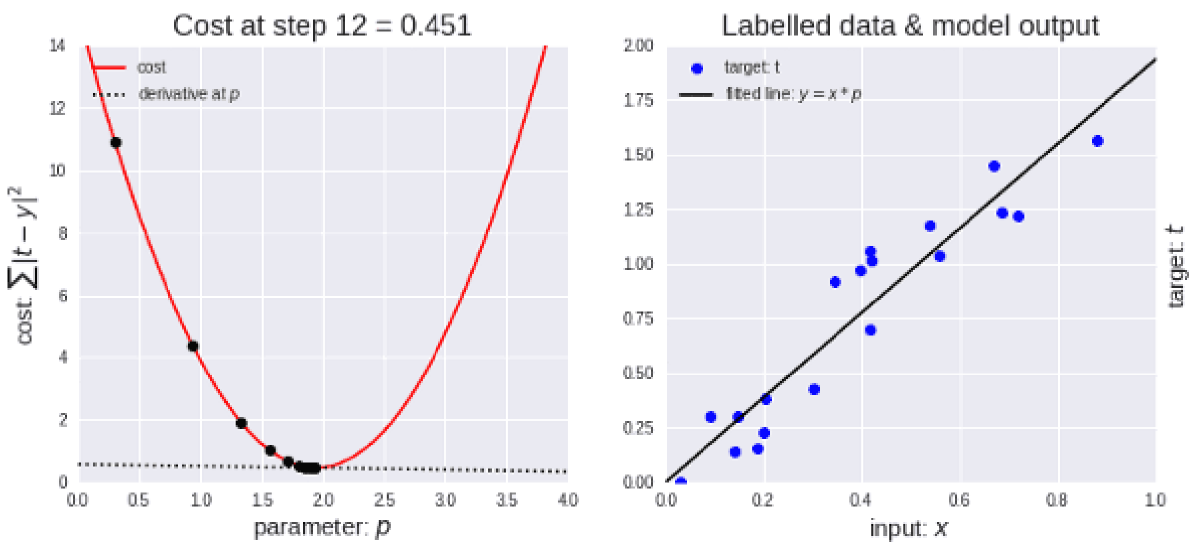
$$\boldsymbol{a}(i+1) = \boldsymbol{a}(i) - \alpha(i) \frac{\partial J_e(\boldsymbol{a}(i))}{\partial \boldsymbol{a}}$$

Adaptação automática

$$\boldsymbol{a}(i+1) = \boldsymbol{a}(i) - \boldsymbol{\alpha}(i) \frac{\partial J_e(\boldsymbol{a}(i))}{\partial \boldsymbol{a}}$$

- Existem várias técnicas de ajuste do valor do passo de aprendizagem durante o treinamento para melhorar o modelo, como:
  - Taxa constante,
  - Decaimento linear e exponencial,
  - Adaptação automática com os algoritmos Adagrad, RMSProp e Adam.
- As duas primeiras usam ***um passo para todos os pesos***, já a última, usa ***um passo independente por peso***.
- Cada técnica visa otimizar a convergência e o desempenho do modelo.

Gradiente descendente



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

$$\boldsymbol{a} \leftarrow \boldsymbol{a} - \alpha \frac{\partial J_e(\boldsymbol{a})}{\partial \boldsymbol{a}}$$ (*eq. de atualização dos pesos*)

- Esse ***processo iterativo de otimização*** que discutimos até agora é chamado de ***gradiente descendente (GD)***.
- Ele está por trás do aprendizado de vários algoritmos de ML: regressão linear, regressão logística, redes neurais em geral, máquinas de vetores de suporte, aprendizado por reforço, etc.
- Como veremos, o GD pode ser implementado de ***3 formas diferentes dependendo de como calculamos o gradiente***.

Versões do gradiente descendente

- Para entendermos as 3 versões do GD, vamos primeiro encontrar o vetor gradiente, $\frac{\partial J_e(\boldsymbol{a})}{\partial \boldsymbol{a}}$, e substituí-lo na equação de atualização dos pesos.
- Considerando o EQM como função de erro e a seguinte função hipótese

$$\hat{y}(n) = \underbrace{a_0 + a_1 x_1(n) + \cdots + a_K x_K(n)}_{\text{Equação do hiperplano}} = \sum_{i=0}^{K} a_i x_i(n) = \boldsymbol{a}^T \boldsymbol{x}(n),$$

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

- Agora podemos encontrar o vetor gradiente.

Versões do gradiente descendente

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

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

$$= -\frac{2}{N}\sum_{n=0}^{N-1}[y(n) - \hat{y}(n)]\boldsymbol{x}(n)^T = -\frac{2}{N}\boldsymbol{X}^T(\boldsymbol{y} - \widehat{\boldsymbol{y}}),$$

onde $\boldsymbol{X}$ é uma matriz $(N \times K + 1)$ com todos os atributos para os N instantes de tempo considerados e $\boldsymbol{y}$ e $\widehat{\boldsymbol{y}}$ são vetores coluna $(N \times 1)$ com todos os valores esperados e de saída da função hipótese para os N instantes de tempo considerados, respectivamente.

- Esse equacionamento pode ser diretamente estendido a polinômios.

Versões do gradiente descendente

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

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

- Podemos ter ***3 versões diferentes, dependendo da quantidade de exemplos***, M, consideradas no somatório acima:
  - ***Gradiente descendente em batelada* (GDB).**
  - ***Gradiente descendente estocástico* (GDE).**
  - ***Gradiente descendente em mini-lotes* (GDML).**

Gradiente descendente em batelada

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

- Utiliza ***todos os exemplos*** do conjunto de treinamento (i.e., $M = N$) para o cálculo do gradiente.
- Pode ser ***computacionalmente complexo*** dependendo do tamanho do modelo e do conjunto de dados.
  - Por processar todos os exemplos, pode ser lento em conjuntos muito grandes e consumir muita memória.
- ***Convergência*** para o mínimo global é ***garantida*** quando a função de erro é ***convexa***.
- É a versão que ***obtém os melhores resultados***.

Gradiente descendente estocástico

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

- Utiliza ***apenas um exemplo*** do conjunto de treinamento (i.e., $M = 1$) para calcular uma ***estimativa estocástica do gradiente***.
  - Versão estocástica pois a cada iteração toma-se uma amostra ***aleatória*** do conjunto para calcular a estimativa do gradiente.
- Quando os ***dados de treinamento são ruidosos***, a ***estimativa*** do gradiente ***é ruidosa***, fazendo com que a ***convergência não ocorra*** ou ***não seja garantida***.
- Entretanto, apresenta ***menor complexidade computacional***, pois é mais rápido e requer menos memória do que o GDB.

Gradiente descendente em mini-lotes

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

- Utiliza um ***subconjunto de exemplos*** do conjunto de treinamento ($M = \mathrm{MB}$) para o cálculo do gradiente.
- Por usar um subconjunto de exemplos, em geral, $1 < \mathrm{MB} < \mathrm{N}$, é mais rápido que o GDB e mais preciso e estável do que o GDE.
- É uma ***generalização*** das duas versões anteriores e a ***versão mais usada no treinamento de redes neurais***.

Exemplo

- Gradiente descendente.

Atividades

- Quiz: “***TP557 – Minimizando o erro***”.
- Exercício: [Gradiente descendente].

Perguntas?

Obrigado!