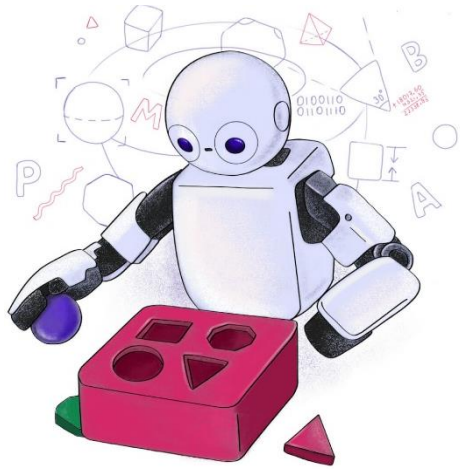
TP557 - Tópicos avançados em IoT e Machine Learning:

Minimizando o erro



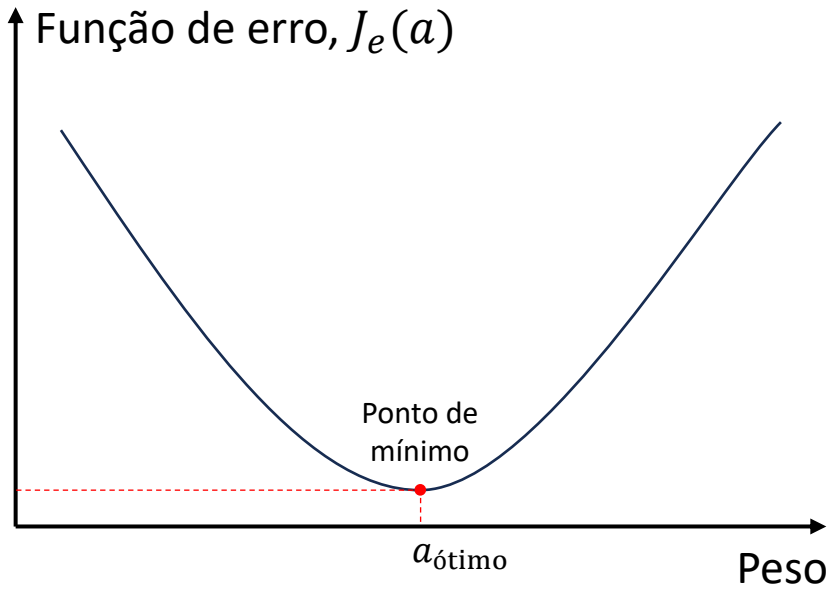
Inatel

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

O que vamos ver?

- Anteriormente, vimos como função 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.
- A ideia é que quanto menor o erro, mais preciso é o seu palpite.
- Portanto, neste tópico, exploraremos como minimizar o erro.

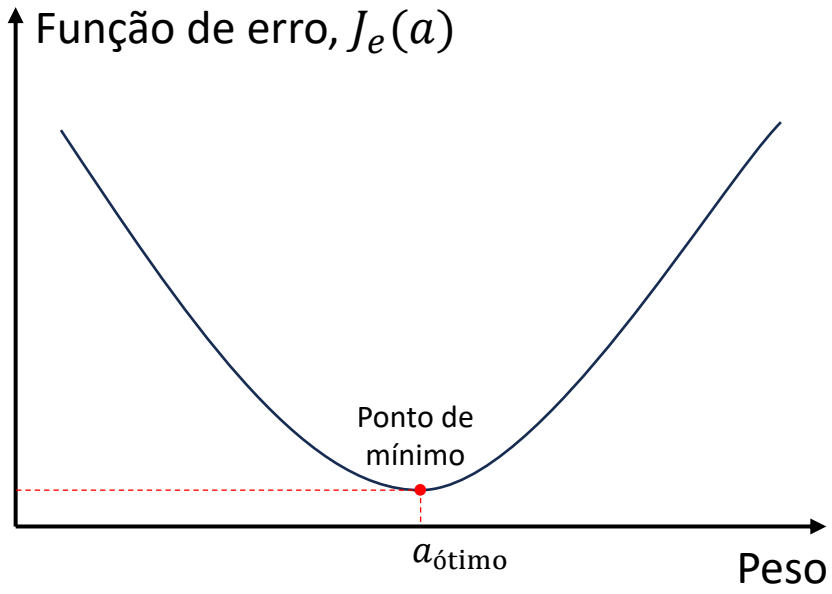
Gradientes e derivadas



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

- Vamos primeiro ver como vetores gradiente e derivadas nos ajudam a minimizar o erro.
- Consequentemente, entenderemos como o algoritmo de otimização/treinamento dos modelos funciona.
- **OBS**.: Vamos usar $J_e(a)$ para definir uma função de erro genérica.

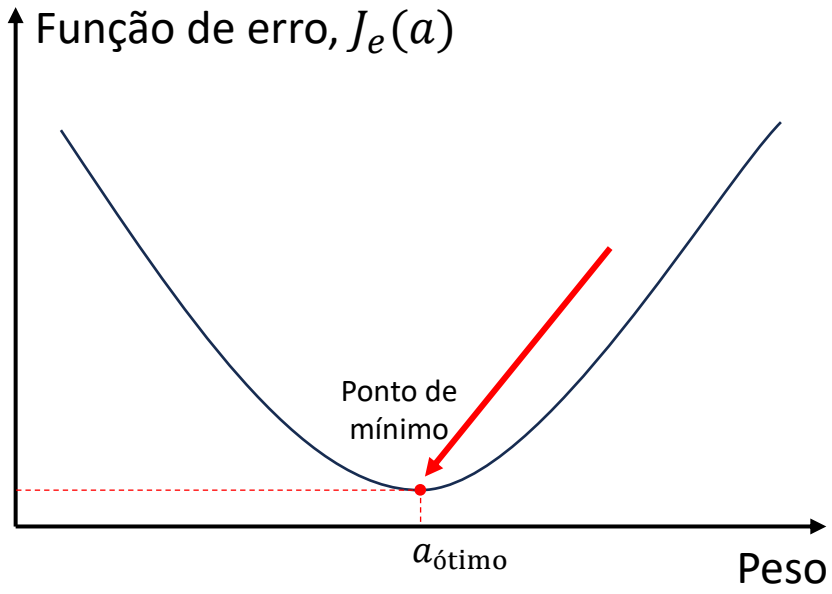
Formato da função de erro



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

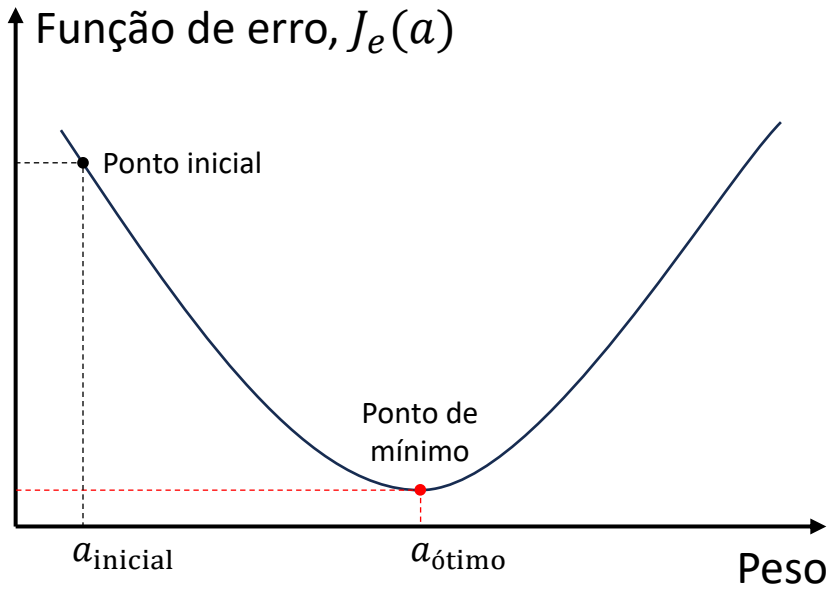
- Lembrem-se que a função de erro que usamos anteriormente, a função do EQM, é quadrática.
- E como vimos no exemplo, funções quadráticas têm a forma de parábolas convexas.
- A convexidade é importante pois garante que a função tenha apenas um ponto de mínimo, o mínimo global.
- Isso permite que encontremos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

Ponto de mínimo



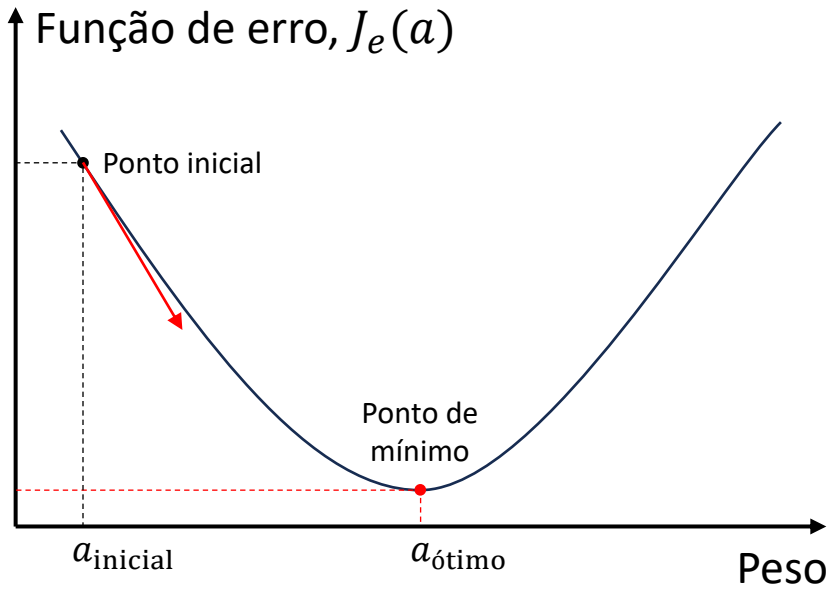
- Se queremos encontrar o ***mínimo*** da função, basta buscarmos a ***parte mais baixa da parábola***.
- Não importa quais sejam os valores dos pesos e onde a função de erro é plotada no gráfico, nós ***sempre teremos certeza de que o mínimo está na parte inferior da parábola***.

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 grande e, consequentemente, saberemos que estamos longe do ponto de mínimo.
- Portanto, quanto menor o erro, mais próximo estaremos do ponto ótimo.

Vetor gradiente



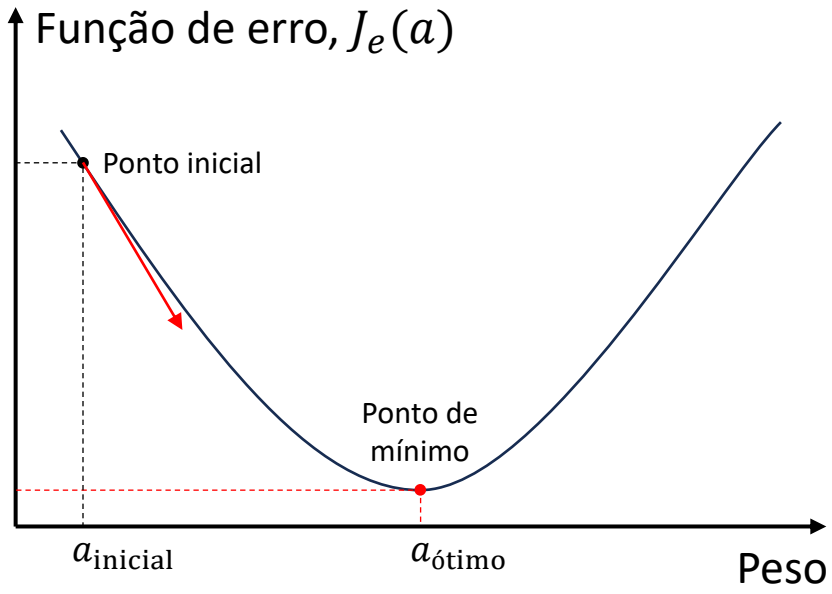
O objetivo é minimizar a função de erro indo na direção indicada pelo gradiente.

- 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 aponta na ***direção de maior crescimento da função*** a partir de um determinado ponto.
- O gradiente pode ser também interpretado como a ***inclinação*** da 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 mínimo***.

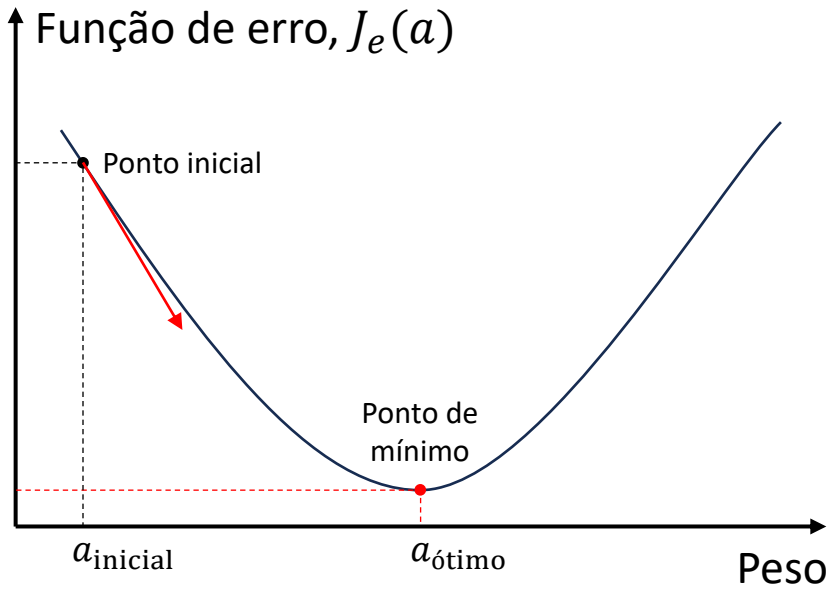
Vetor gradiente



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

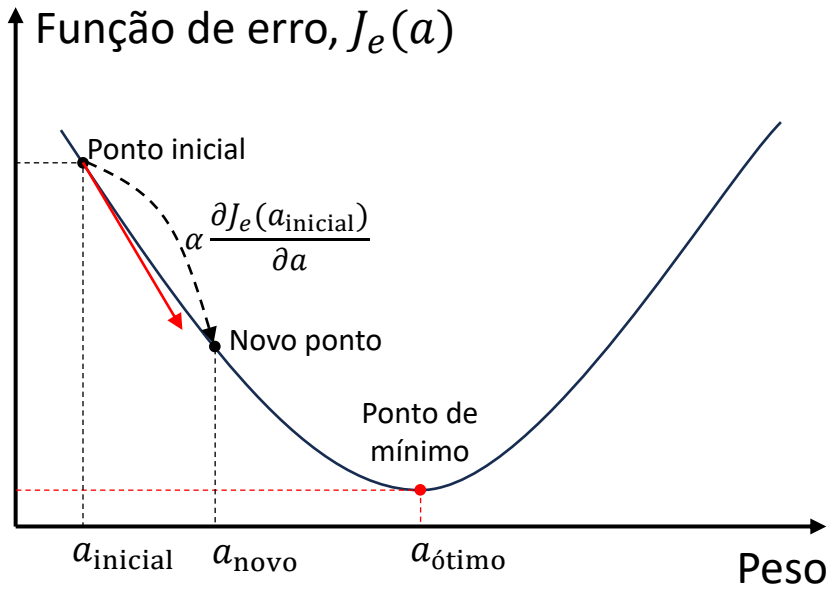
- Porém, queremos o mínimo da função, o que fazer?
- Basta irmos na ***direção oposta*** a do gradiente (negativo do gradiente), a qual ***aponta para a direção de maior decréscimo da função*** a partir do ponto.

Vetor gradiente



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

Passo de aprendizagem

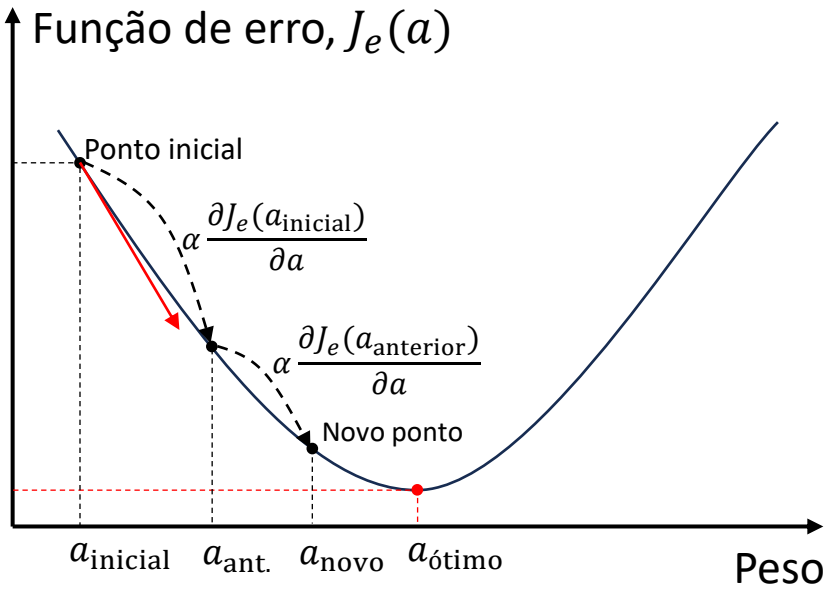


$$a_{\text{novo}} = a_{\text{inicial}} - \alpha \frac{\partial J_e(a_{\text{inicial}})}{\partial a}$$

sentido aposto ao apontado pelo gradiente

- Portanto, se quisermos ir para o ponto de mínimo a partir de um ponto qualquer, podemos dar um ***passo*** na direção apontada pelo gradiente.
- Nós sabemos a direção e podemos escolher o ***tamanho do passo*** para darmos naquela direção.
- O ***tamanho do passo*** é frequentemente chamado de ***taxa ou passo de aprendizagem*** e é, normalmente, denotado por α.
- Vejam que atualizamos o peso atual usando uma fração do gradiente.

Otimização iterativa

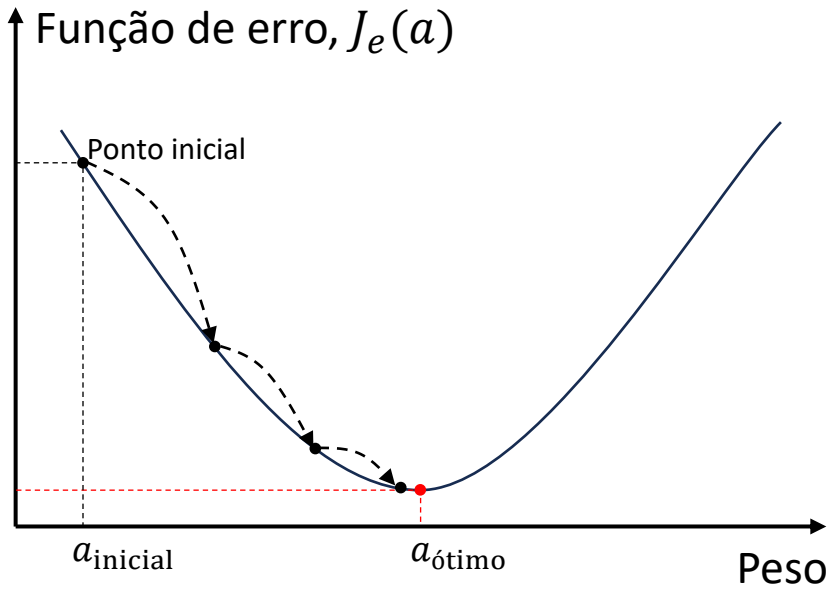


$$a_{\text{novo}} = a_{\text{anterior}} - \alpha \frac{\partial J_e(a_{\text{anterior}})}{\partial a}$$

Equação de atualização dos pesos

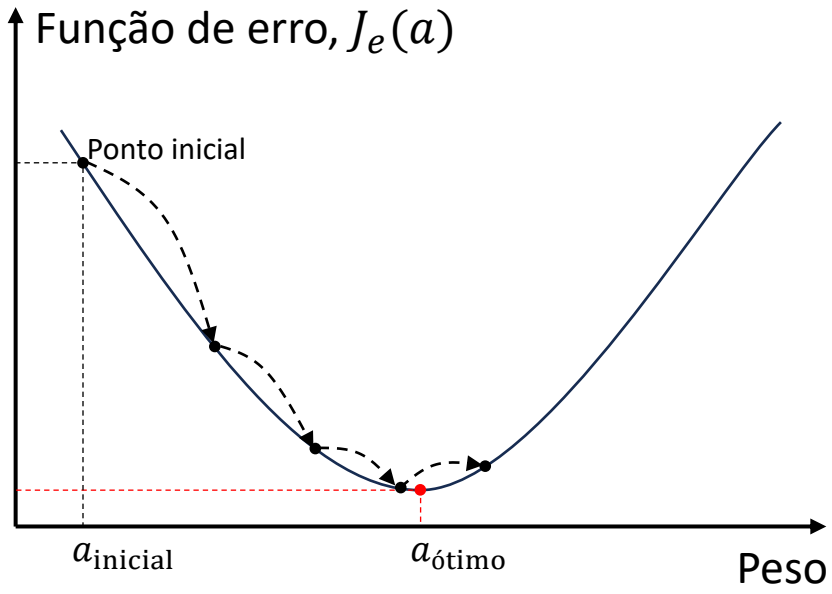
- Portanto, dada a direção do gradiente e um passo de aprendizagem, agora podemos ***iterativamente*** dar passos em direção ao ponto de mínimo.
- A cada ***iteração*** calculamos o gradiente no ponto atual, atualizamos os pesos com uma fração do gradiente e calculamos o gradiente no novo ponto.
- Repetimos esse processo até que a inclinação da reta tangente ao ponto 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



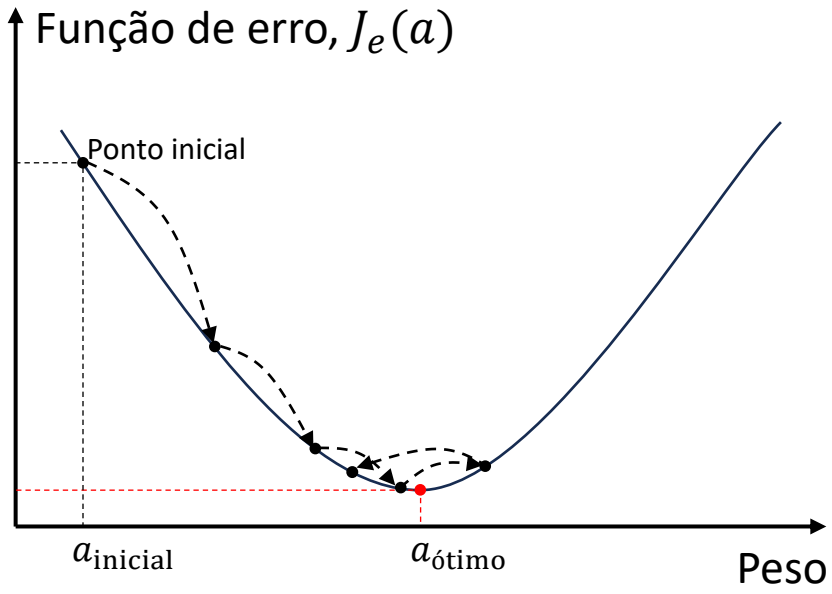
- Mais uma iteração e podemos nos mover para mais perto ainda do ponto de mínimo, $a_{\text{ótimo}}$.
- Porém, devemos tomar cuidado, com o tamanho do passo de aprendizagem.

Tamanho do passo de aprendizagem



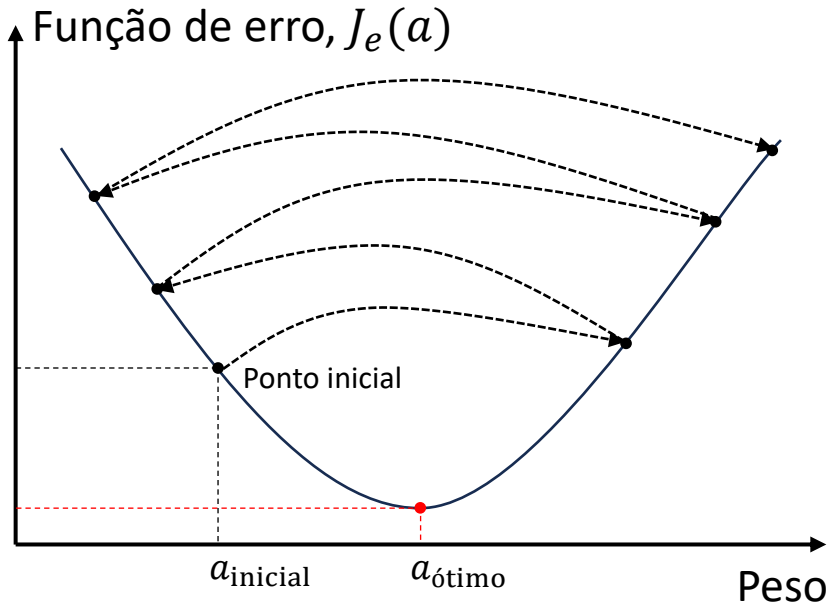
- Se o passo de aprendizagem for muito grande, podemos ultrapassar o ponto de mínimo.

Tamanho do passo de aprendizagem



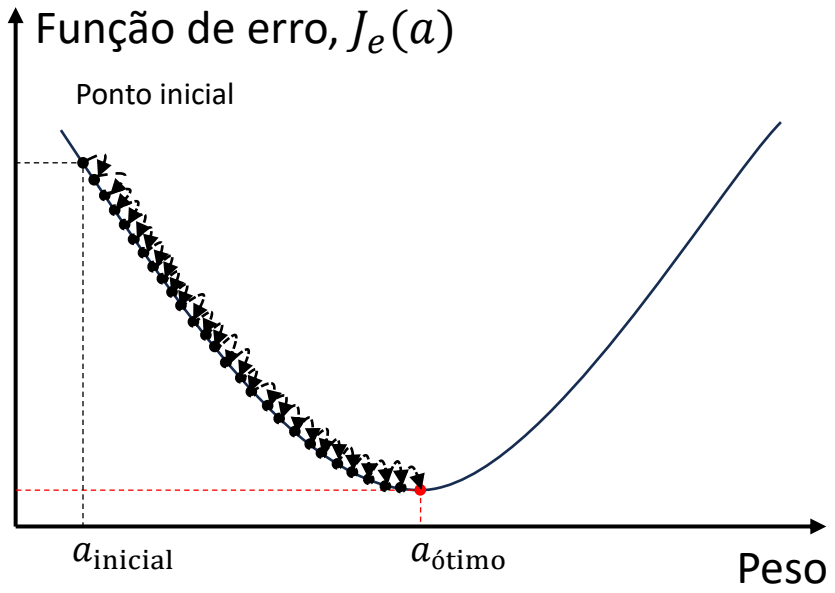
- Se o passo de aprendizagem for muito grande, o processo de otimização pode ficar ziguezagueando de um lado para o outro do fundo da função sem nunca atingir o ponto de mínimo.

Tamanho do passo de aprendizagem



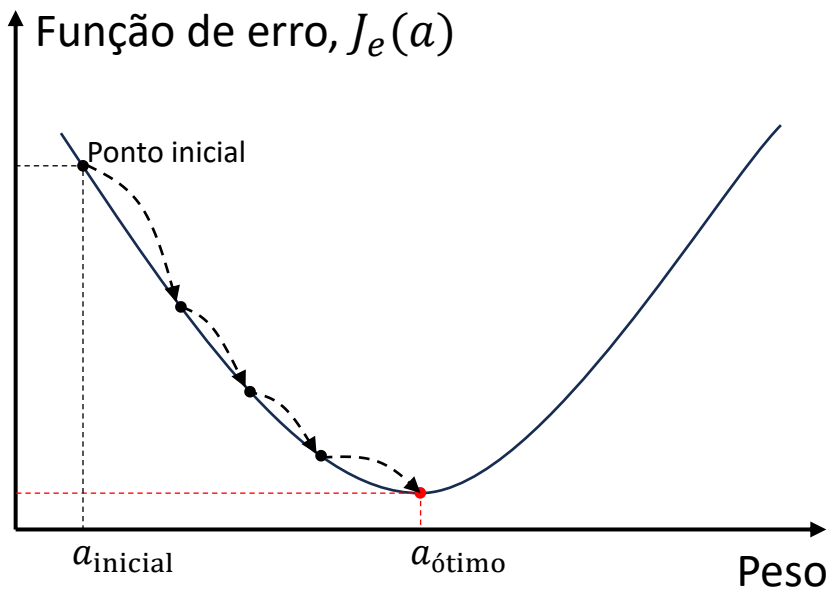
- Dependendo do quão grande for o valor do passo de aprendizagem, pode ocorrer até a divergência ao invés da convergência.
- Se isso ocorrer, após algumas iterações, ocorre o estouro da precisão numérica das variáveis.

Tamanho do passo de aprendizagem



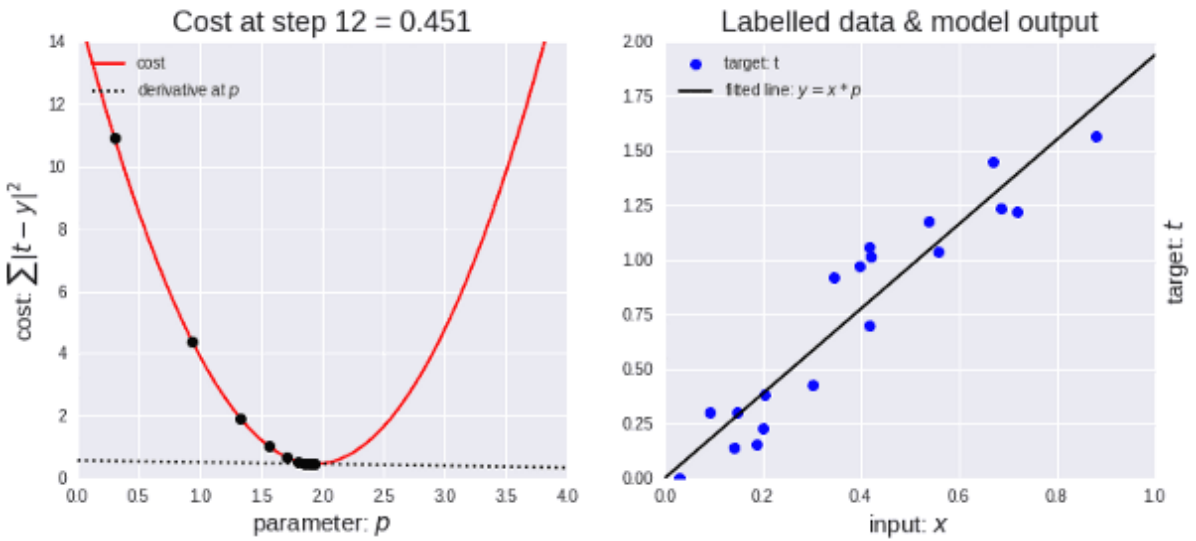
- Outra questão, menos problemática que passos grandes, é a situação oposta.
- Passos muito pequenos fazem com que se leve muito tempo, i.e., iterações, para atingir o ponto de mínimo.
- Se cada iteração levar um tempo razoável para ser executada, o tempo necessário para se atingir o ponto de mínimo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

Tamanho do passo de aprendizagem



- 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.

Gradiente descendente



$\boldsymbol{a}$ ← 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 otimiz***ação que discutimos até agora é chamado de ***gradiente descendente (GD)***.
- Ele está por trás de vários algoritmos de ML: regressão linear, regressão logística, redes neurais em geral, máquinas de vetores de suporte, aprendizado por reforço, etc.
- O GD pode ser implementado de 3 formas diferentes.

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

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

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*** é dado por

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

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

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 $\hat{\boldsymbol{y}}$ 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 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 implementações diferentes, dependendo da quantidade de amostras, M, consideradas no somatório acima:
  - $M = N \rightarrow$ **Gradiente descendente em batelada (GDB)**: é computacionalmente complexo dependendo do tamanho do modelo e do conjunto de dados, porém é a versão que obtém os melhores resultados.
  - $M = 1 \rightarrow$ ***Gradiente descendente estocástico (GDE)***: é rápido por usar uma ***estimativa do gradiente***, a qual pode ser ruidosa, fazendo com que a convergência não exista ou não seja garantida.
  - $M = \mathrm{MB} \rightarrow$ ***Gradiente descendente em mini-lotes***: por usar um pequeno grupo de amostras, 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***.

Atividades

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

Perguntas?

Obrigado!