

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



Inatel

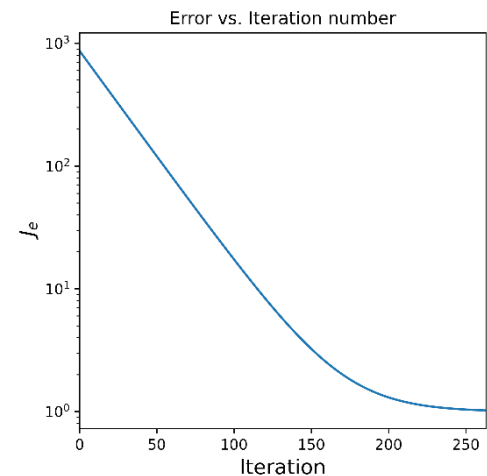
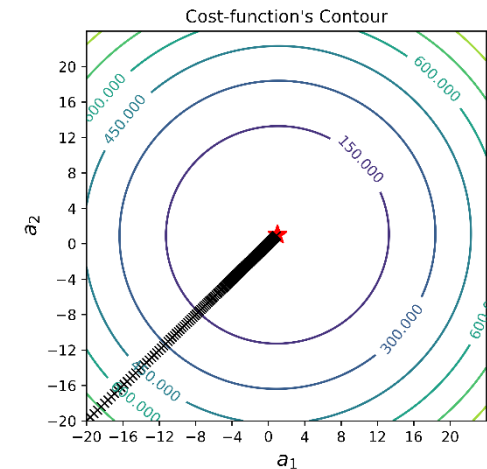
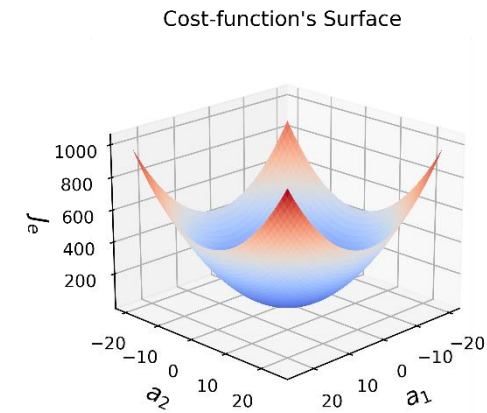
Felipe Augusto Pereira de Figueiredo
felipe.figueiredo@inatel.br

Recapitulando

- Discutimos sobre o vetor gradiente.
- Aprendemos dois algoritmos que usam o vetor gradiente para a resolução de problemas de otimização.
- Vimos as três versões do gradiente descendente, como implementá-las em Python e as comparamos.
- Nesta parte, discutiremos o quão importante é o ajuste do passo de aprendizagem, α .

Escolha do Passo de Aprendizagem

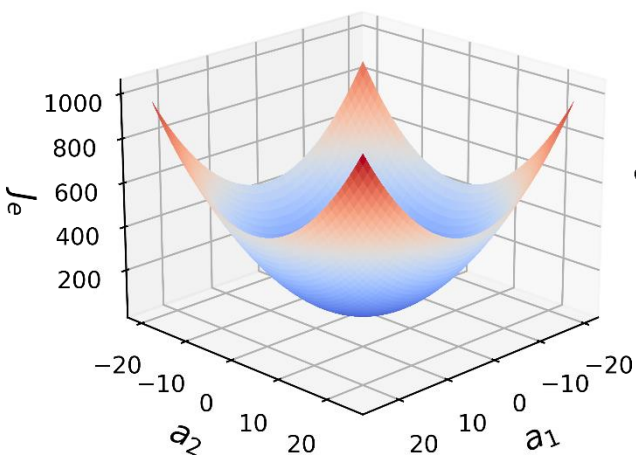
- Conforme nós já aprendemos, enquanto o **sentido** e a **direção** para o mínimo são determinados pelo **vetor gradiente** da **função de erro**, o **passo de aprendizagem** determina o quão grande esse passo é dado naquela direção e sentido.
- Portanto, a **escolha do passo de aprendizagem é muito importante**:
 - Caso ele seja muito pequeno, a convergência do algoritmo levará muito tempo.
 - **Exemplo**: com $\alpha = 0.01$ atinge o valor ótimo após mais de 250 épocas.
 - Passos muito curtos, fazem com que o algoritmo caminhe vagarosamente em direção ao **mínimo global** da **função de erro**.



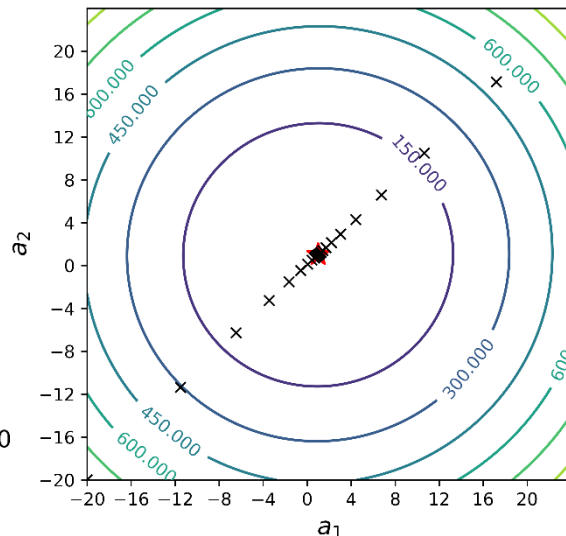
Escolha do Passo de Aprendizagem

- Caso o ***passo de aprendizagem*** seja muito grande, o algoritmo pode nunca convergir.
- O algoritmo fica “pulando” ou “oscilando” de um lado para o outro do vale até que converge, por sorte.
- Em alguns casos, a cada iteração o algoritmo “pula” para um valor mais alto que antes, e assim, divergindo.

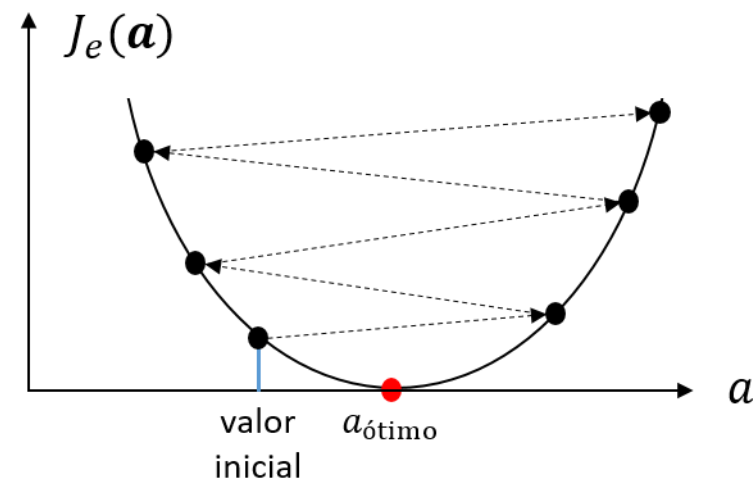
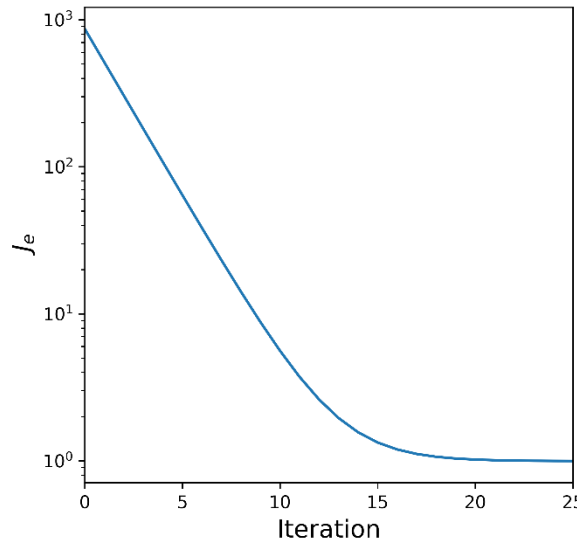
Cost-function's Surface



Cost-function's Contour

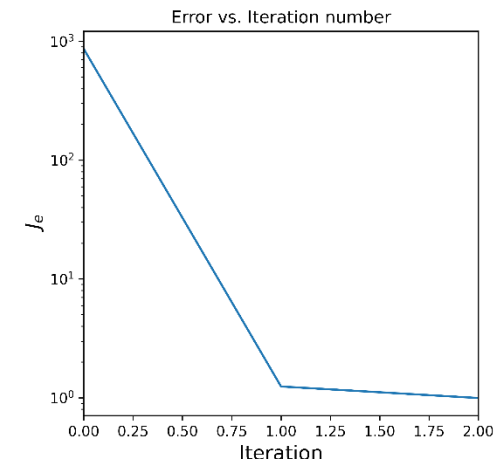
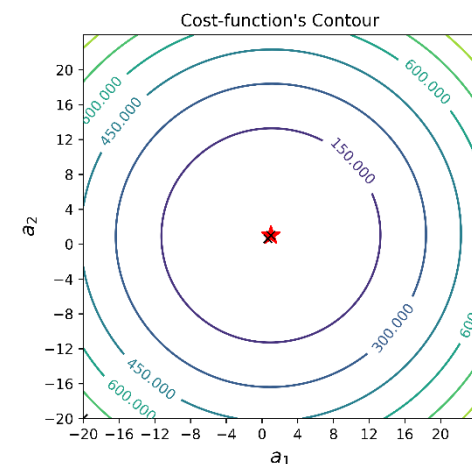
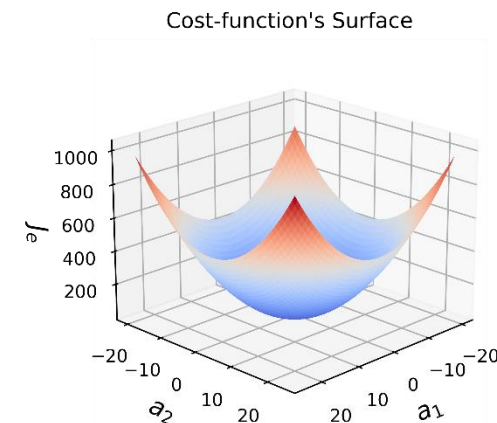


Error vs. Iteration number



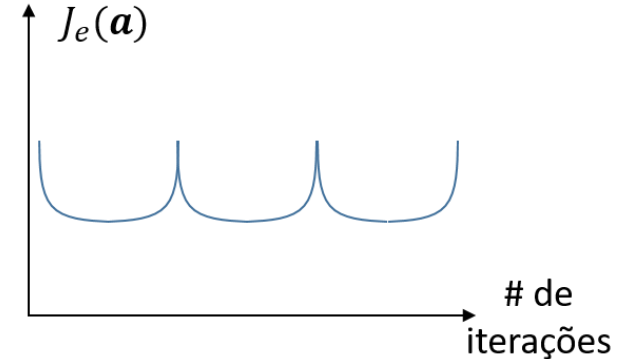
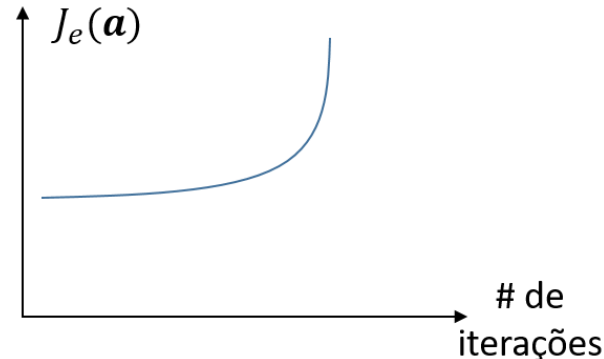
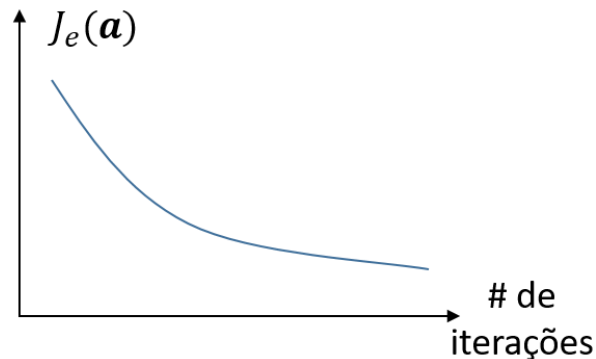
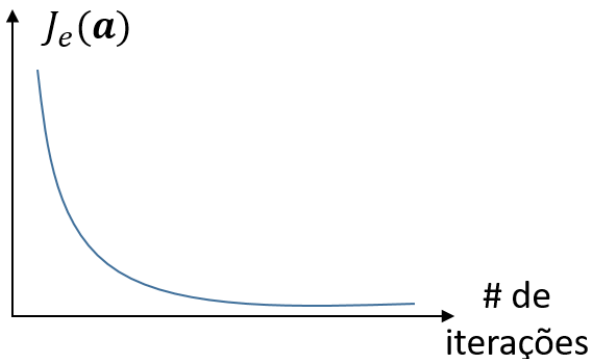
Escolha do Passo de Aprendizagem

- Portanto, o valor ***passo de aprendizagem*** deve ser ***experimentado/explorado*** para se encontrar um ***valor ótimo*** que acelere a ***descida do gradiente*** de forma ***estável*** (ou seja, acelere a convergência).
- O exemplo ao lado, converge para o ***mínimo global*** em apenas 2 iterações.
- Portanto, escolher o passo de aprendizagem é muitas vezes desafiador e demorado.
 - Passos muito grandes fazem com que o algoritmo aprenda rápido demais ao custo de um modelo final que seja sub-ótimo ou que o treinamento divirja ou se torne instável (oscilação).
 - Passos muito pequenos resultam num longo treinamento, podendo o algoritmo, por exemplo, ficar preso em um mínimo local ou mesmo nunca atingir um mínimo.

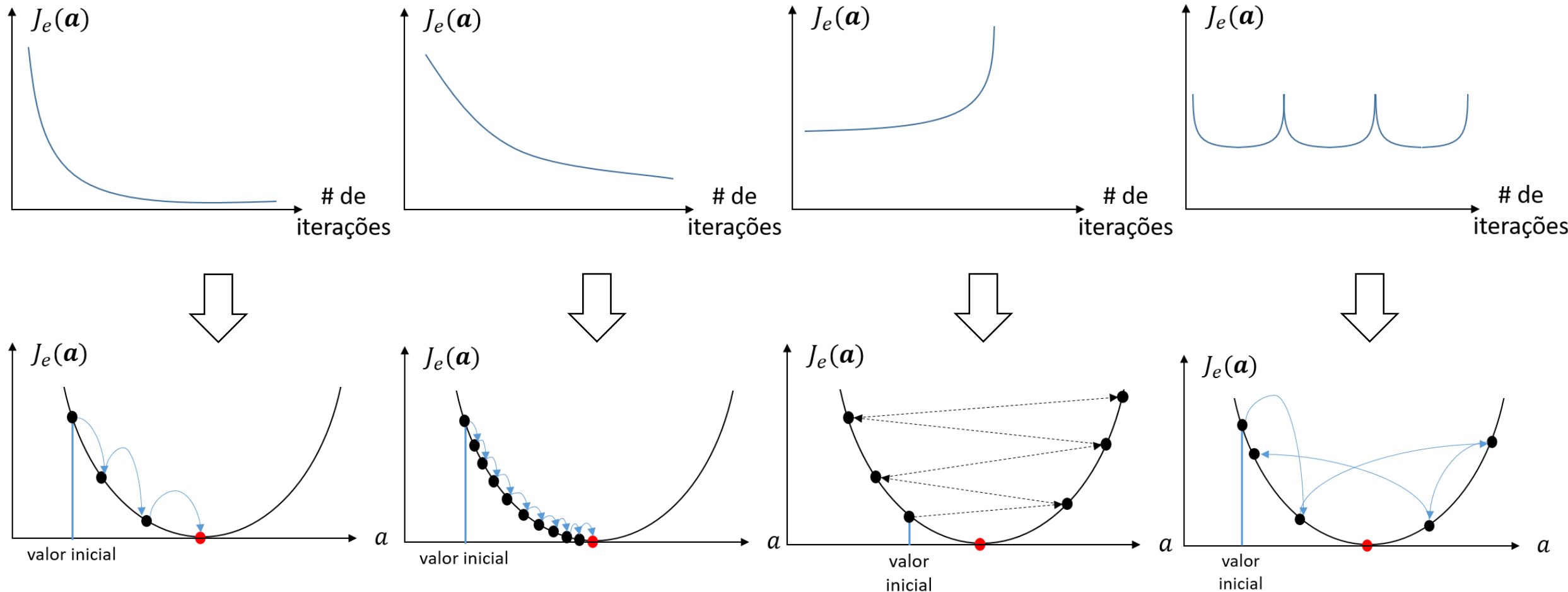


Como depurar o algoritmo do GD?

- Uma das maneiras de se **depurar** (principalmente quando não é possível se plotar o gráfico da superfície de contorno) o algoritmo do **gradiente descendente** é plotar o gráfico do erro (EQM) em função do número de iterações ou épocas.
 - Figura A \Rightarrow Passo ótimo: converge rapidamente
 - Erro diminui rapidamente nas primeiras épocas e depois diminui quase que a uma taxa constante.
 - Convergência pode ser declarada quando o erro entre duas épocas subsequentes for menor do que um limiar pré-definido (e.g., $1e-3$).
 - Figura B \Rightarrow Passo pequeno demais: convergência lenta.
 - Figuras C e D \Rightarrow Passo grande demais: divergência.

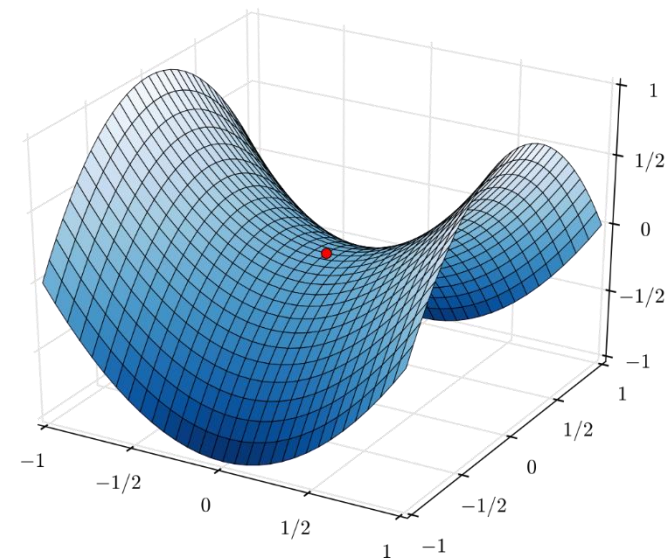
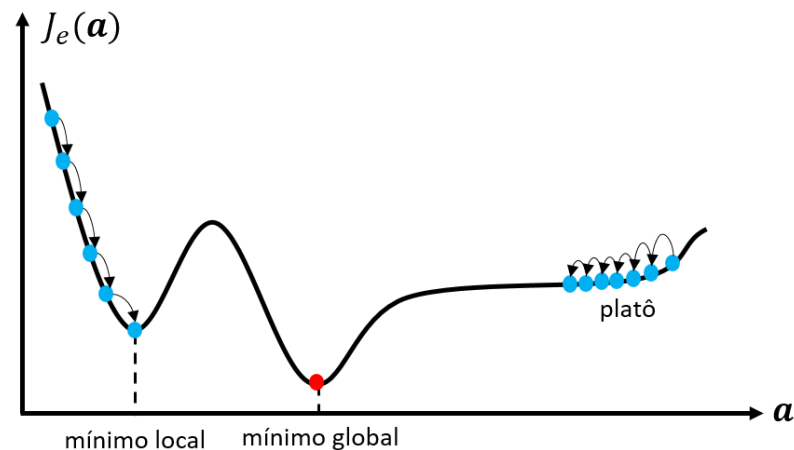


Como depurar o algoritmo do GD?



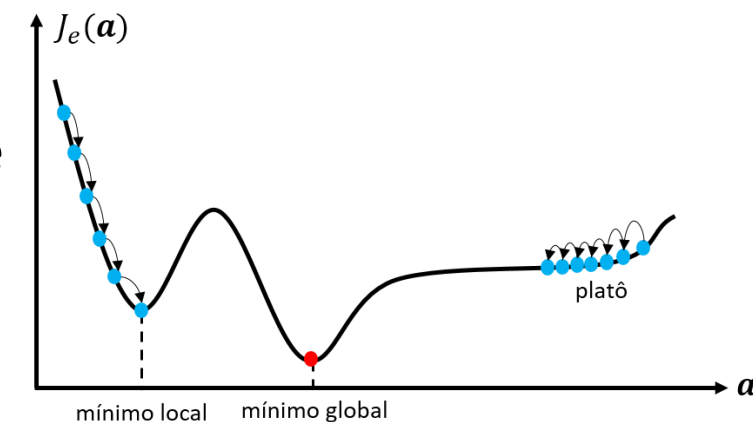
Desafios Encontrados pelo Gradiente Descendente

- Como vimos, as superfícies de **funções de erro** que utilizam o **erro quadrático médio** para treinamento de modelos de regressão linear são sempre **convexas**, se assemelhando a um vale ou a uma tigela.
 - **Implicações:** não existem mínimos locais, apenas um mínimo global. É também uma função contínua com uma inclinação que nunca muda abruptamente.
 - **Consequência:** o gradiente descendente **garantidamente** se aproxima do mínimo global (dado que você espere tempo suficiente e que o **passo de aprendizagem** não seja grande demais).
- Porém, nem todas as **superfícies de erro** se parecem com vales ou tigelas, ou seja, são convexas. Algumas podem ter vários mínimos locais, platôs, pontos de sela, e todo tipo de “**terreno irregular**”, dificultando a convergência para o mínimo global. Redes neurais são exemplos de modelos com funções de erro não-convexas e bem irregulares.
- Além disso, como vimos antes, passos grandes podem desestabilizar o algoritmo e passos muito pequenos aumentar demais o tempo de convergência.



Desafios Encontrados pelo Gradiente Descendente

- A figura mostra dois dos principais desafios encontrados pelo **algoritmo do gradiente descendente**. Se a inicialização aleatória dos pesos iniciar o algoritmo à
 - esquerda, ele convergirá para um mínimo local, que não é tão bom quanto o mínimo global.
 - direita, levará muito tempo para atravessar o platô (gradiente próximo de zero pois a inclinação é próxima de 0 graus) e, se ele parar muito cedo, nunca alcançará o mínimo global.
- Dado que o passo de aprendizagem seja grande o suficiente, como garantir que o mínimo encontrado é o global e não um mínimo local?
 - O que se faz é treinar o modelo várias vezes, sempre inicializando os pesos aleatoriamente, com a esperança de que em alguma dessas vezes ele inicialize mais próximo do mínimo global.



Tarefas

- **Quiz:** “*T319 - Quiz - Regressão: Parte III (1S2021)*” que se encontra no MS Teams.
- **Exercício número 1 do [Laboratório #4](#).**
 - Pode ser baixado do MS Teams ou do GitHub.
 - Pode ser respondido através do link acima (na nuvem) ou localmente.
 - [Instruções para resolução e entrega dos laboratórios](#).

Como configurar o passo de aprendizagem?

As abordagens abaixo são as mais usadas para se configurar o passo de aprendizagem.

- **Ajuste manual:** envolve a escolha de valores através de tentativa e erro.
- **Redução programada:** variação do passo de aprendizagem ao longo do processo de treinamento. A forma mais simples é diminuir o passo de aprendizagem linearmente de um grande valor inicial para um pequeno valor.
- **Momentum:** adiciona a média do histórico de atualizações (i.e., o termo momentum) à atualização corrente dos pesos. O termo momentum tem o efeito de suavizar o processo de aprendizado, tornando as atualizações menos ruidosas.
- **Variação adaptativa:** o algoritmo de aprendizado monitora o desempenho do modelo no conjunto de treinamento e, conseqüentemente, o passo de aprendizagem pode ser ajustado em resposta ao desempenho. Pode ajustar os pesos de cada elemento do vetor gradiente de forma independente. Por exemplo, se o algoritmo ficar preso em um platô, pode-se aumentar o passo.

Ajuste Manual do Passo de Aprendizagem

- Em geral, não é possível se calcular a priori o melhor passo de aprendizagem.
- Portanto, uma primeira abordagem para se ajustar o passo é através do **ajuste manual**, onde sua configuração é feita através de **tentativa e erro** (como fizemos antes) até que um passo bom o suficiente seja encontrado.
 - A desvantagem é que isso pode levar muito tempo, imagine um conjunto de treinamento com milhões de exemplos e dezenas ou até mesmo centenas de pesos a serem ajustados e você ter que esperar até que o treinamento termine para saber se um determinado passo é bom o suficiente.
- Uma forma alternativa consiste em realizar uma análise de sensibilidade do passo de aprendizagem para o modelo escolhido, também chamada de **busca em grade** (do Inglês, **grid search**).
 - **Grid search** é um método de pesquisa exaustiva que faz a busca em um subconjunto do espaço de hiperparâmetros, especificado manualmente, de um algoritmo de aprendizagem.
- **Grid search** nos ajuda a encontrar uma ordem de magnitude onde podem residir bons passos de aprendizagem.
- Típicamente, uma **busca em grade** envolve a escolha de valores em uma escala logarítmica, por exemplo, $\{1, 10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}\}$.
- Depois de escolhido, o valor do passo é sempre constante.

Exemplo: Ajuste Manual com Grid Search

```
# Define the number of examples.
N = 1000

# Features.
x1 = np.random.randn(N, 1)
x2 = np.random.randn(N, 1)

# Observable function.
y_noisy = x1 + x2 + np.random.randn(N, 1)

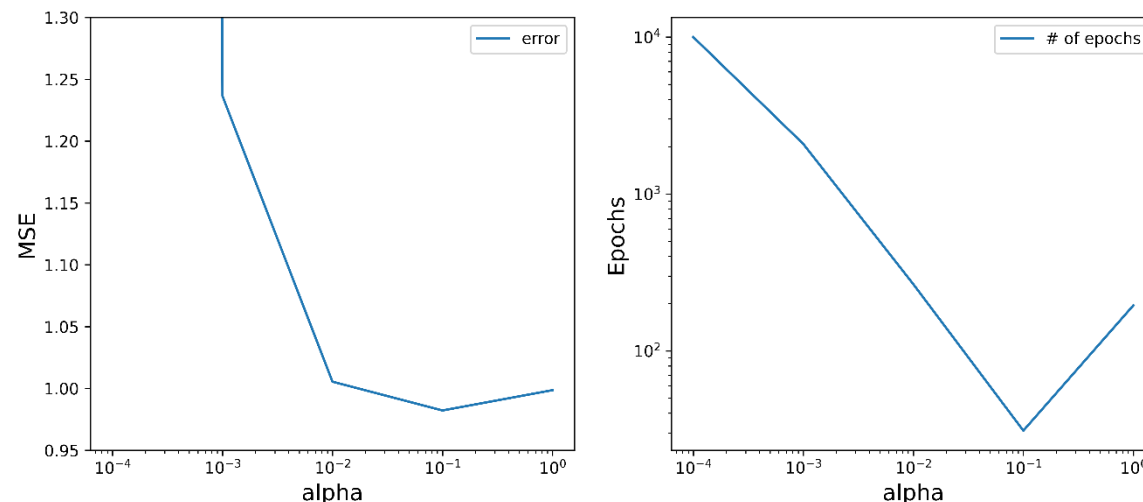
# Maximum number of epochs.
maxEpochs = 10000

# Logarithmic search.
alphas = [0.0001, 0.001, 0.01, 0.1, 1.0]

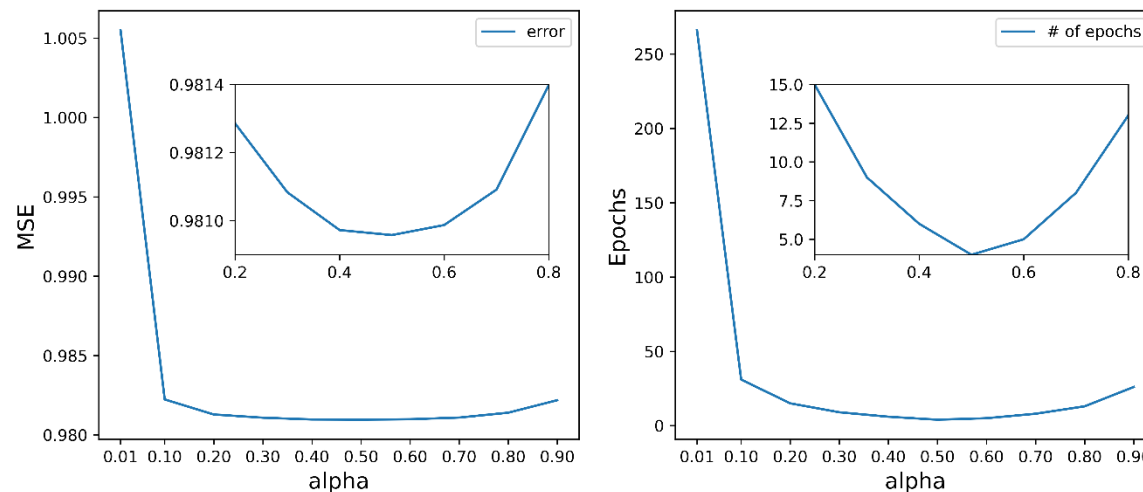
for i in range(0, len(alphas)):
    gd = MyGD(alphas[i], maxEpochs, [-20.0, -20.0])
    gd.fit(X, y_noisy)
    scores.append(gd.score(X, y_noisy))
    iterations.append(gd.iteration)

# Fine-tuning the learning rate.
alphas = [0.01, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]

for i in range(0, len(alphas)):
    gd = MyGD(alphas[i], maxEpochs, [-20.0, -20.0])
    gd.fit(X, y_noisy)
    scores.append(gd.score(X, y_noisy))
    iterations.append(gd.iteration)
```



Ajuste fine em torno
do melhor valor de α



[Exemplo: linear_regression_grid_search.ipynb](#)

Ajuste por Redução Programada do Passo de Aprendizagem

- Os três tipos mais comuns de implementação da **redução programada** do passo de aprendizagem são:
 - **Decaimento gradual**: também conhecido como **decaimento por etapas** ou **por degraus**. Ele reduz a taxa de aprendizagem de um fator α a cada número pré-definido de iterações ou épocas, β . Um valor típico para reduzir a taxa de aprendizado é de $\alpha = 0.5$ a cada número pré-definido de épocas.
 - **Decaimento exponencial**: tem a forma matemática $\alpha = \alpha_0 e^{-kt}$, onde α_0 e k são hiperparâmetros e t é o número da iteração (pode-se se usar também o número de épocas).
 - **Decaimento $1/t$** : ou **temporal**, tem a forma matemática $\alpha = \alpha_0 / (1+kt)$ onde α_0 e k são hiperparâmetros e t é o número da iteração.
- Na prática, o **decaimento gradual** é mais o utilizado entre os 3, pois seus **hiperparâmetros** (a fração de decaimento e os intervalos de tempo para redução) são mais interpretáveis do que o hiperparâmetro k , que dita a taxa de decaimento do passo de aprendizagem.
- Existem outros esquemas de redução programada de α , mas todos são extensões de algum destes 3.

Exemplo: GDE com Redução Programada de α

```
import numpy as np
```

```
# Define the number of examples.
```

```
N = 1000
```

```
# Generate target function.
```

```
x1 = np.random.randn(N, 1)
```

```
x2 = np.random.randn(N, 1)
```

```
y = x1 + x2 + np.random.randn(N, 1)
```

```
# Concatenate both column vectors, x1 and x2.
```

```
X = np.c_[x1, x2]
```

```
# Number of epochs.
```

```
n_epochs = 1
```

```
# Initial learning rate.
```

```
alpha_int = 0.1
```

```
# Learning schedule function.
```

```
def learning_schedule(alpha_int, t):
```

```
    drop = 0.5
```

```
    epochs_drop = 4.0
```

```
    alpha = alpha_int * math.pow(drop, math.floor((1+t)/epochs_drop))
```

```
    return alpha
```

```
a = np.random.randn(2,1)
```

```
# Stochastic gradient-descent loop.
```

```
for epoch in range(n_epochs):
```

```
    si = random.sample(range(0, N), N)
```

```
    for i in range(N):
```

```
        random_index = si[i]
```

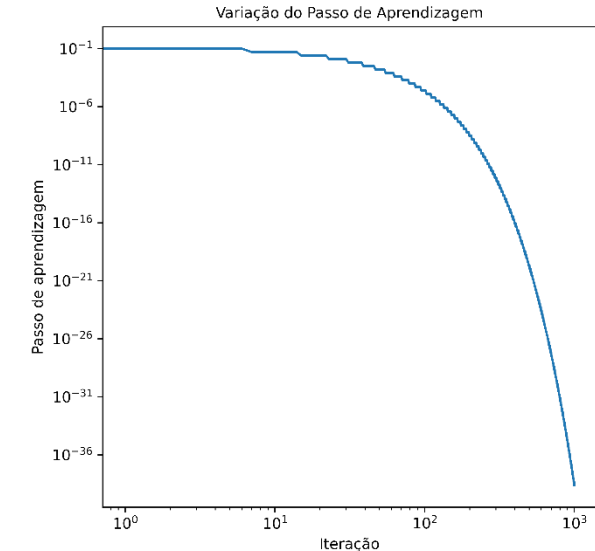
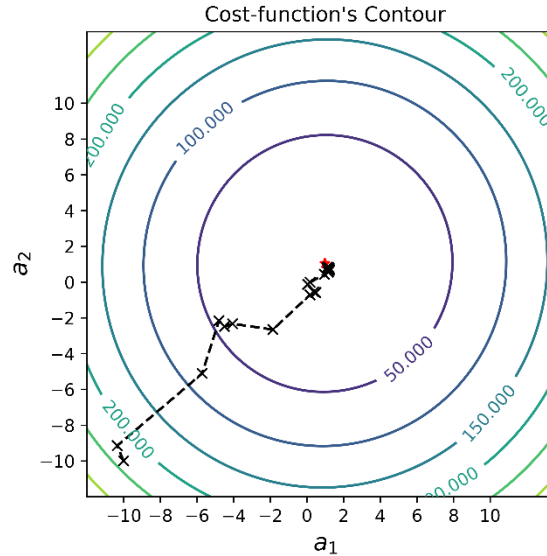
```
        xi = X[random_index:random_index+1]
```

```
        yi = y[random_index:random_index+1]
```

```
        gradient = -2*xi.T.dot(yi - xi.dot(a))
```

```
        alpha = learning_schedule(alpha_int, k, epoch*N + i)
```

```
        a = a - alpha * gradient
```

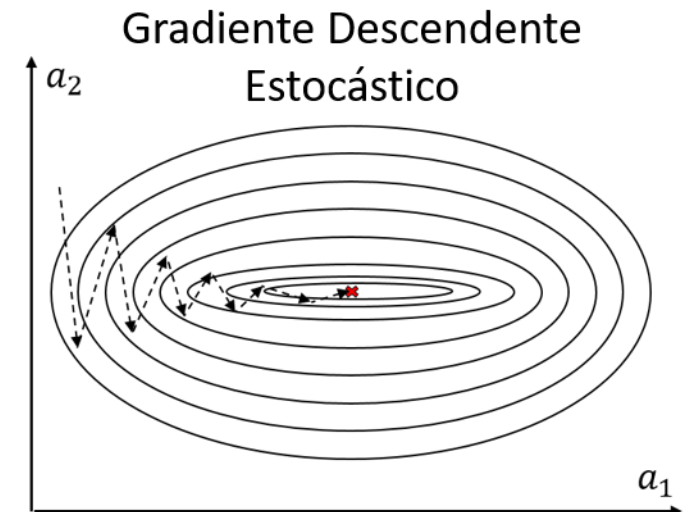


[Exemplo: stochastic gradient descent with learning schedule and with figures.ipynb](#)

- Exemplo com esquema de variação do passo conhecido como “decaimento gradual”
- O caminho também não é direto/regular para o mínimo.
- Apresenta algumas mudanças de direção e sentido ao longo do caminho.
- Oscilação em torno do mínimo é bastante minimizada pelo esquema de variação do passo.
- Os passos começam com grandes valores e depois diminuem cada vez mais, permitindo que o algoritmo se estabilize próximo ao mínimo global.

Ajuste do Passo de Aprendizagem com Termo Momentum

- O GDE aproxima o gradiente com apenas um exemplo fazendo com que as derivadas parciais se tornem "*ruidosas*", ou seja, a direção que o algoritmo toma à caminho do mínimo varia a cada atualização dos pesos.
- Esse problema dos gradientes "*ruidosos*" é mais acentuado em superfícies que se assemelham a **ravinas**, que são áreas onde as curvas da superfície são muito mais abruptas em uma dimensão do que em outra.
- O GDE tem problemas para caminhar em tais superfícies.
- Nessas superfícies, o GDE oscila ao longo das encostas da **ravina** (movimento de zig-zag), enquanto faz um progresso lento e hesitante em direção ao mínimo global, como na figura ao lado.
- Como poderíamos mitigar esse problema?
- O **algoritmo do momentum** é um esquema de ajuste do passo de aprendizagem, α , simples e que ajuda a acelerar a convergência do GDE e amortece as oscilações.
- O algoritmo introduz uma variável \mathbf{v} que desempenha o papel da **velocidade**, ou seja, é a direção e a rapidez com que os parâmetros se movem através do **espaço de parâmetros**.

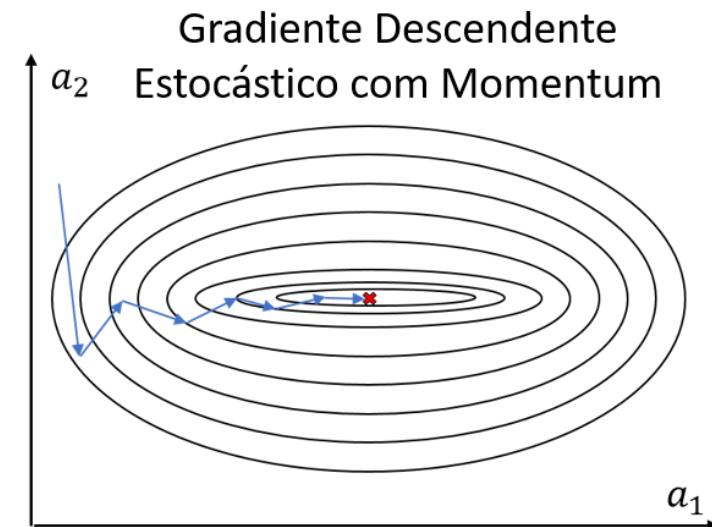
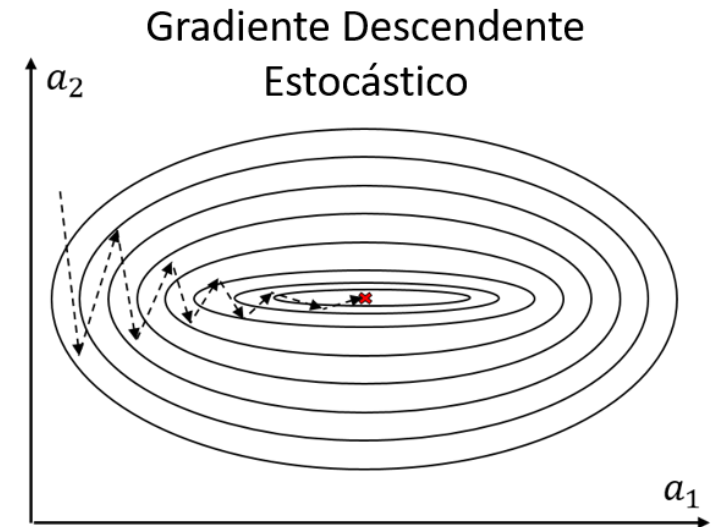


Ajuste do Passo de Aprendizagem com Termo Momentum

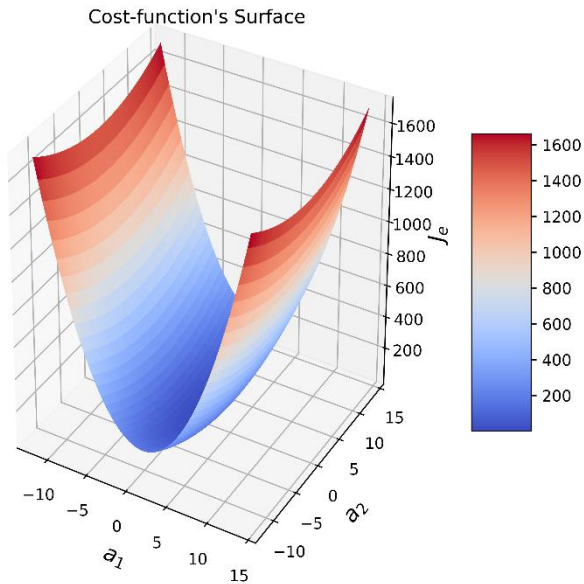
- O algoritmo acelera a convergência e amortece oscilações adicionando uma fração, μ (**chamado de coeficiente de momentum**), do vetor de atualização dos pesos da iteração anterior ao vetor de atualização atual:

$$\mathbf{v} \leftarrow \mu \mathbf{v} - \alpha \frac{\partial (y - \hat{y})^2}{\partial \mathbf{a}}$$
$$\mathbf{a} \leftarrow \mathbf{a} + \mathbf{v}$$

- Quanto maior for o valor de μ , maior será a influência de gradientes anteriores na direção atual.
- O algoritmo acumula uma **média móvel exponencialmente decrescente de gradientes anteriores** e, portanto, pode fornecer uma estimativa melhor, que está mais próxima da derivada parcial real do que os cálculos “ruidosos”.
- Ao analisarmos a equação de atualização dos pesos, vemos que a variável \mathbf{v} faz com que a atualização seja maior para dimensões cujos gradientes apontam nas mesmas direções e reduz atualizações para dimensões cujos gradientes mudam de direção. Como resultado, temos convergência mais rápida e oscilação reduzida.



Exemplo: GDE com Termo Momentum



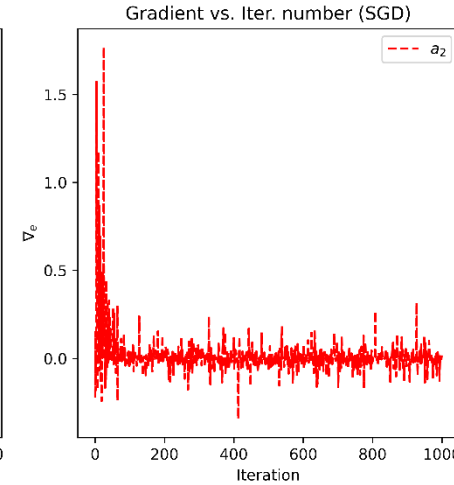
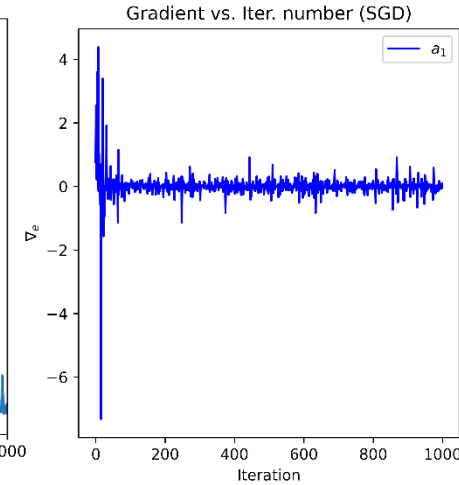
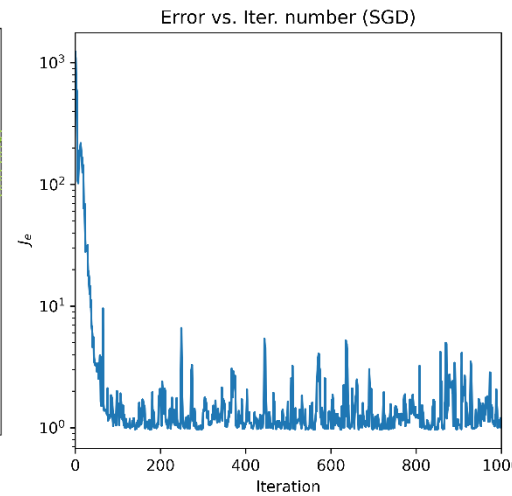
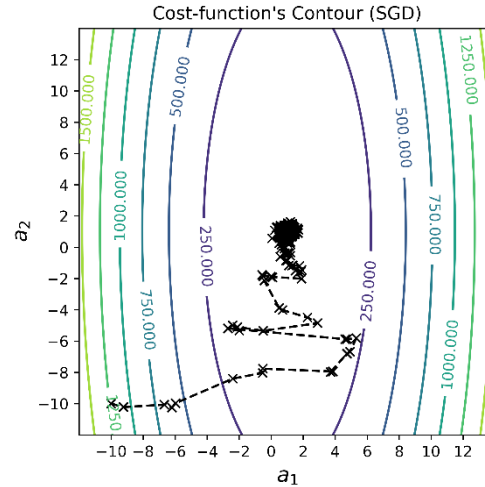
Função observável

$x_1 = 3.0 * \text{randn}(N, 1)$

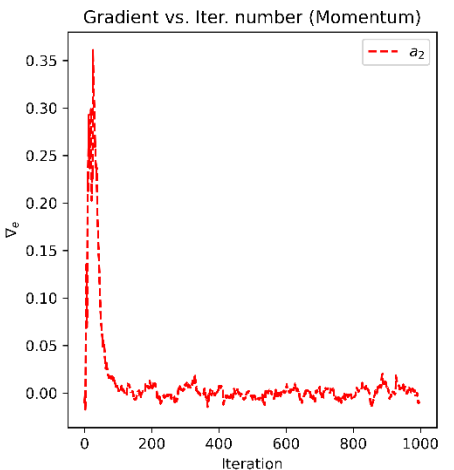
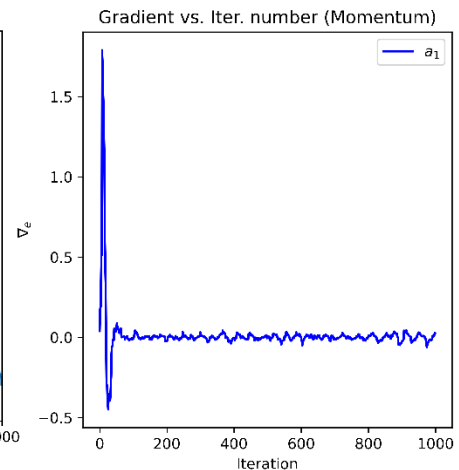
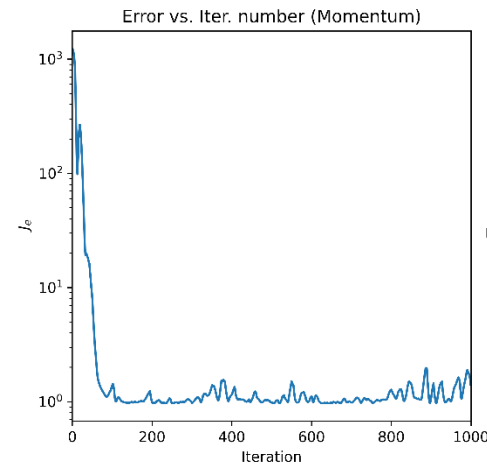
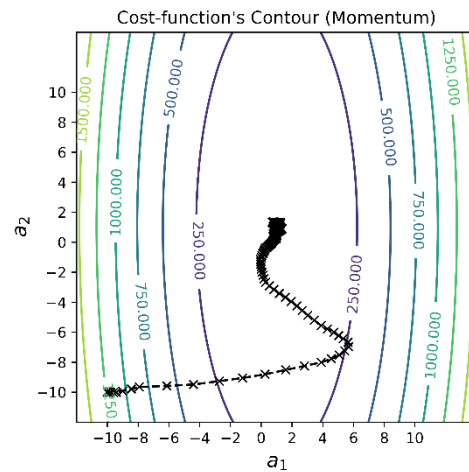
$x_2 = \text{randn}(N, 1)$

$y = x_1 + x_2 + \text{randn}(N, 1)$

$\alpha = 0.02$



$\alpha = 0.001, \mu = 0.9$



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

Ajuste do Passo de Aprendizagem por Variação Adaptativa

- Na **variação adaptativa**, o passo é adaptativamente ajustado de acordo com a performance do modelo além disso, pode ter passos diferentes para cada peso do modelo e os atualiza independentemente.
 - Os passos são atualizados de acordo com valores obtidos pelo modelo
 - Por exemplo, enquanto o desempenho estiver aumentando, o passo é mantido constante. Quando o desempenho se estabiliza, diminui-se o passo. Alternativamente, o passo de aprendizagem pode ser aumentado se o desempenho não melhorar por um número fixo de iterações.
 - Na maioria dos casos, não é necessário se ajustar manualmente nenhum hiperparâmetro como no caso dos esquemas de redução programada.
 - E quando existe algum hiperparâmetro a ser ajustado o esquema normalmente funciona muito bem para uma grande gama de valores.
 - **Exemplos:** Adam, Adagrad, RMSprop, etc.

Implementação: GDE com Scikit-Learn

- A biblioteca **Scikit Learn** disponibiliza a classe **SGDRegressor** para realizar regressão linear utilizando o Gradiente Descendente Estocástico.
- A classe possui vários parâmetros que podem ser configurados (tipo de função de erro, esquema de variação do passo de aprendizagem, etc.).
- A **função de erro** pode ser configurada entre várias opções, mas por padrão, a classe usa o **erro quadrático médio**.
- É possível definir o **esquema de variação do passo de aprendizagem**: constante, redução programada ou adaptativo.
- Por padrão o esquema é o da escala inversa, “**invscaling**”
$$\alpha = \frac{\alpha_{init}}{i^{power}}$$
- Onde α_{init} é o passo inicial (por padrão = 0.01), i é o número da iteração e $power$ é o expoente da escala inversa (por padrão = 0.25).
- Os outros tipos de GD não são implementados pela biblioteca.

```
import numpy as np
```

```
# Usamos a classe SGDRegressor do módulo Linear da biblioteca sklearn.  
from sklearn.linear_model import SGDRegressor
```

```
# Número de exemplos
```

```
N = 1000
```

```
# Criamos os features e labels.
```

```
x1 = np.random.randn(N, 1)
```

```
x2 = np.random.randn(N, 1)
```

```
y = 2*x1 + 4*x2 + np.random.randn(N, 1)
```

```
# Concatena os vetores coluna x1 e x2.
```

```
X = np.c_[x1, x2]
```

```
# Instancia a classe SGDRegressor.
```

```
sgd_reg = SGDRegressor(max_iter=50, fit_intercept=False)
```

```
# Treina o modelo.
```

```
sgd_reg.fit(X, y.ravel())
```

```
print('a1: %1.4f' % (sgd_reg.coef_[0]))
```

```
print('a2: %1.4f' % (sgd_reg.coef_[1]))
```

```
a1: 1.9844
```

```
a2: 3.9802
```

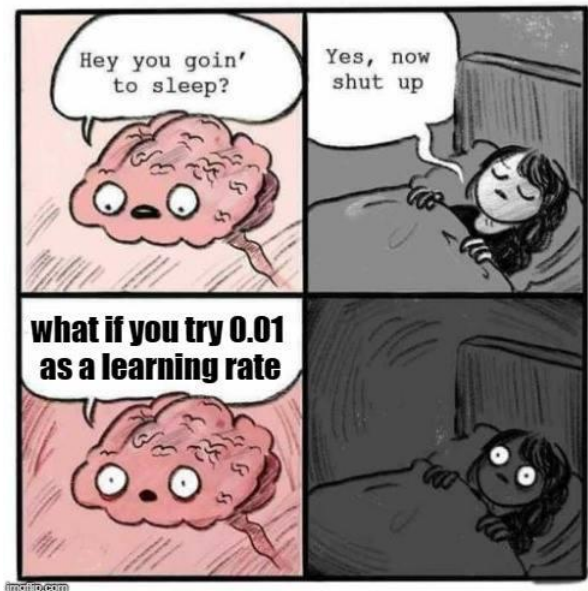


[Exemplo: SGD with scikit learn lib.ipynb](#)

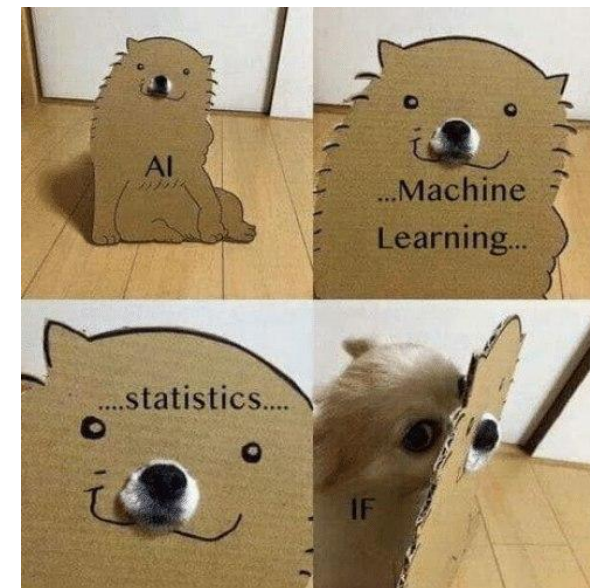
Tarefas

- **Quiz:** “*T319 - Quiz - Regressão: Parte IV (1S2021)*” que se encontra no MS Teams.
- **Exercícios 2 e 3 do [Laboratório #4](#).**
 - Pode ser baixado do MS Teams ou do GitHub.
 - Pode ser respondido através do link acima (na nuvem) ou localmente.
 - [Instruções para resolução e entrega dos laboratórios](#).

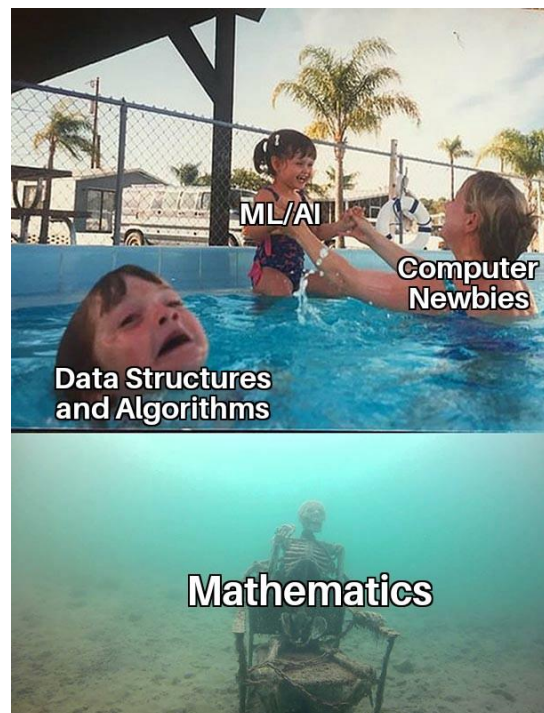
Obrigado!



When someone asks why you never stops talking about machine learning

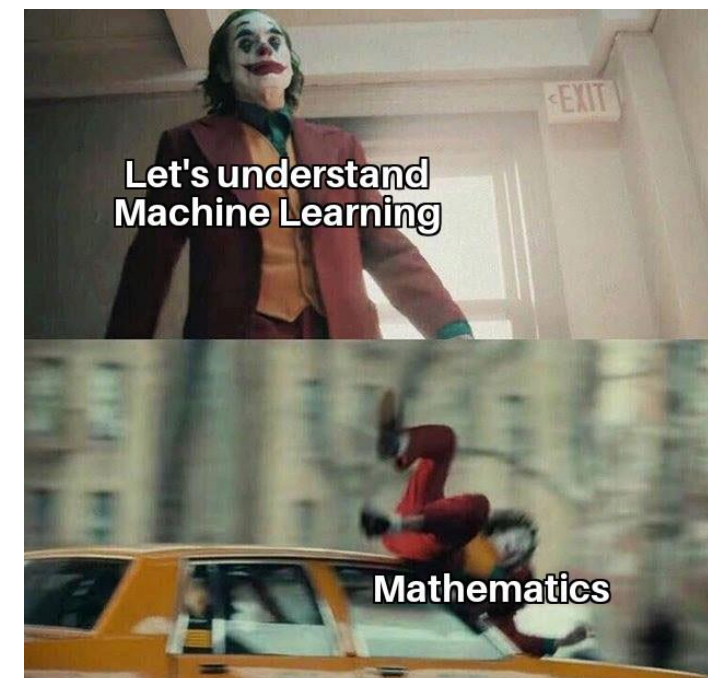


IF IF IF IF IF IF IF IF IF IF WE!

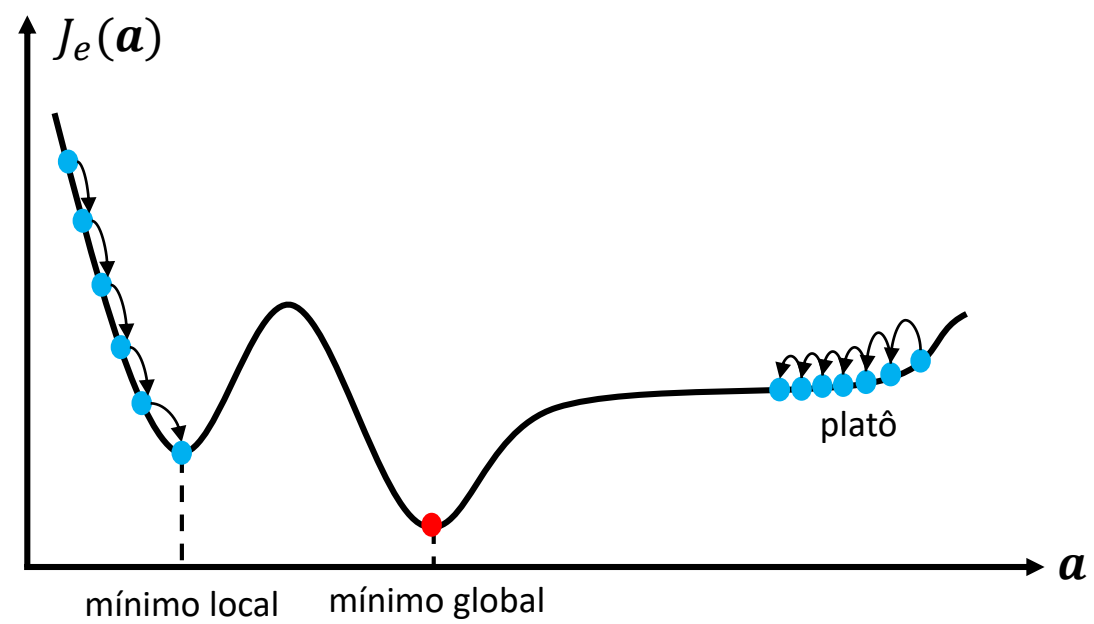


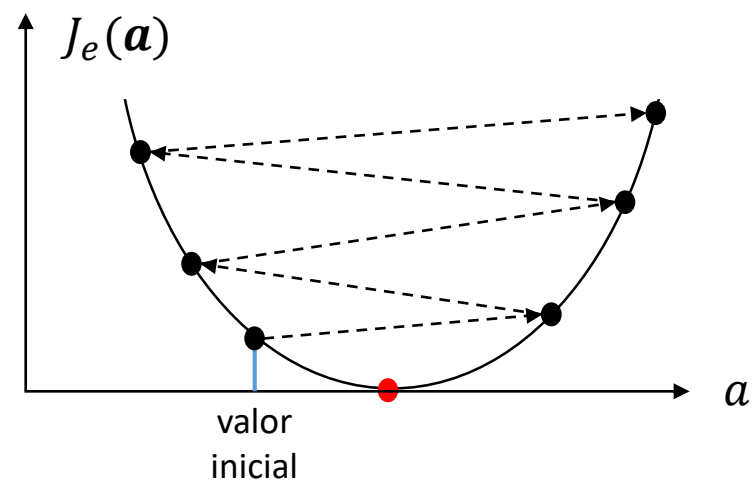
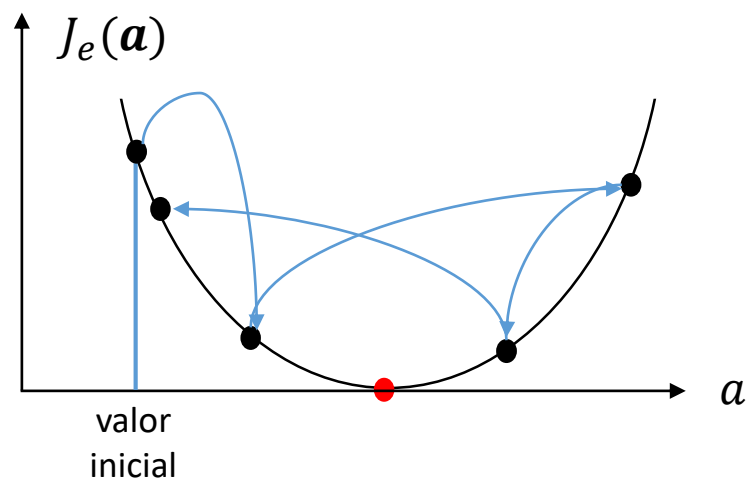
Albert Einstein: Insanity Is Doing the Same Thing Over and Over Again and Expecting Different Results

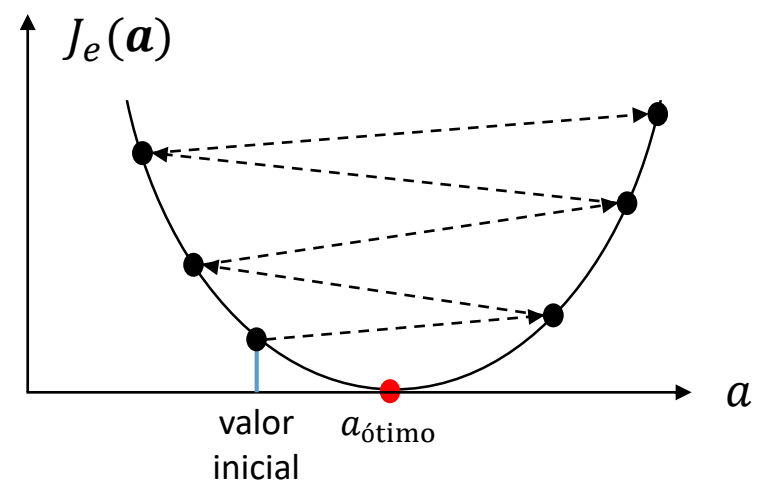
Machine learning:

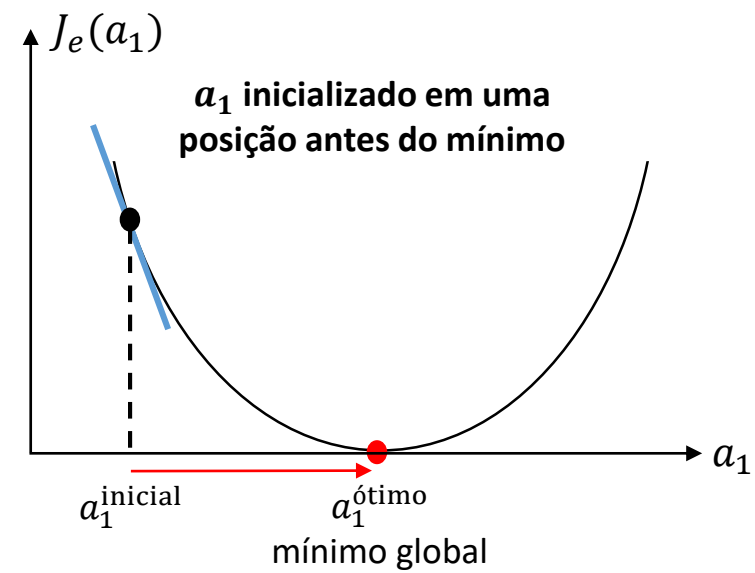


FIGURAS

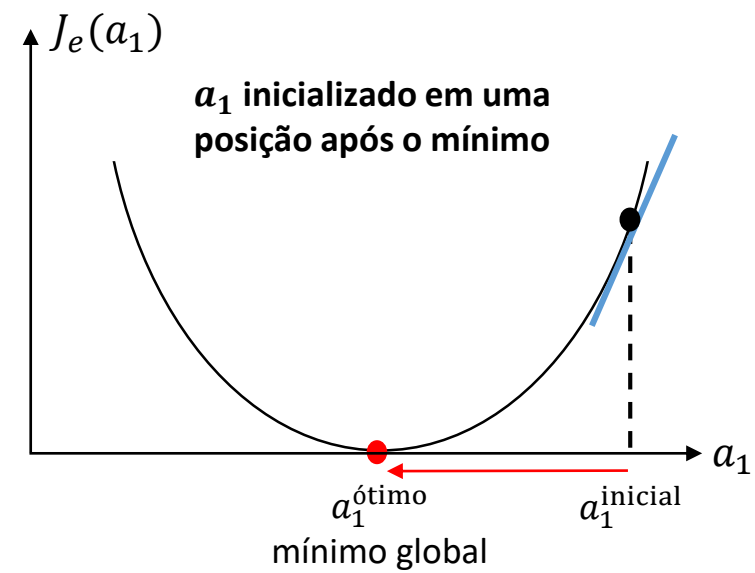




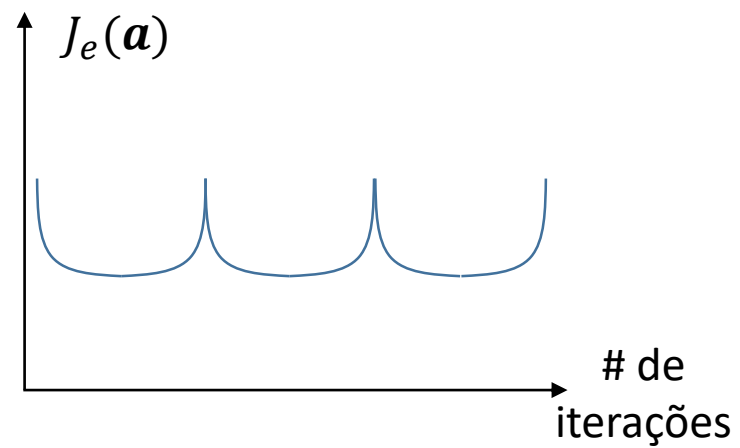
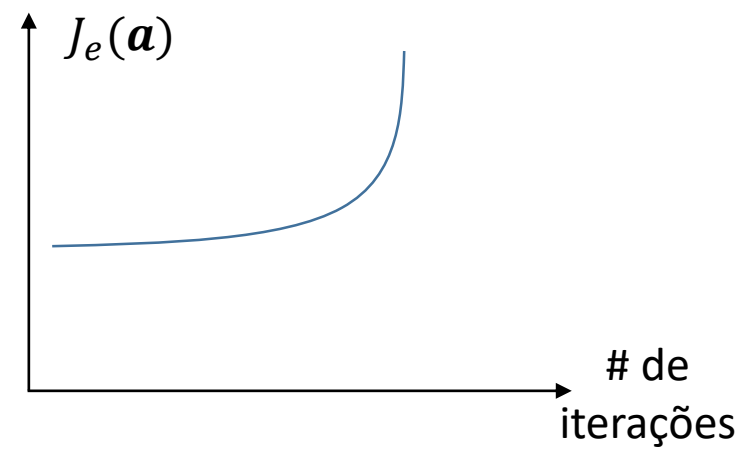
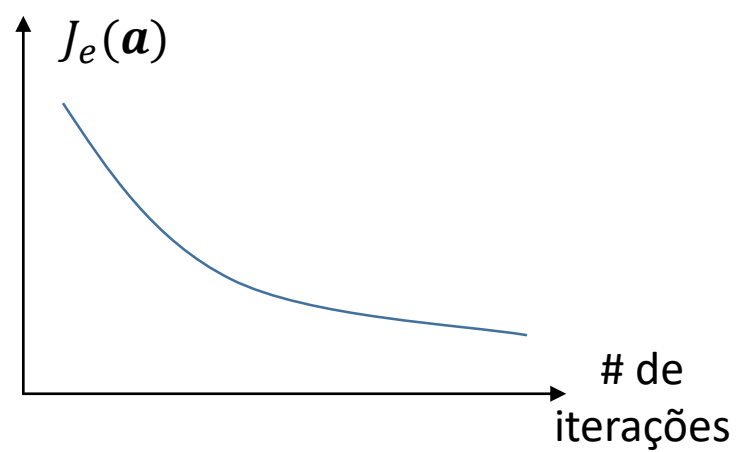
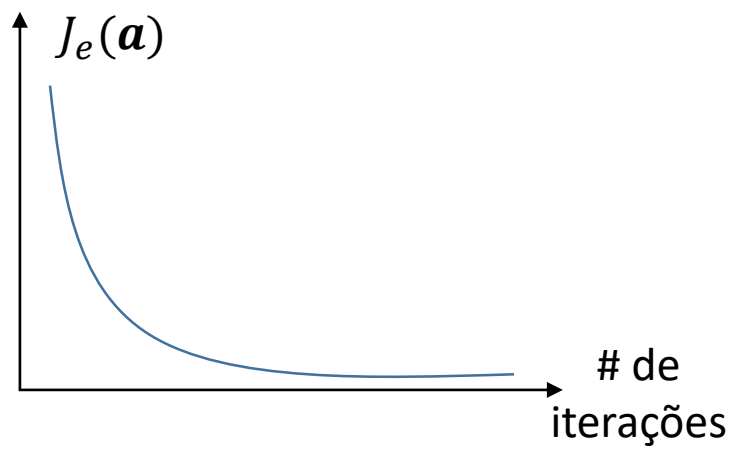


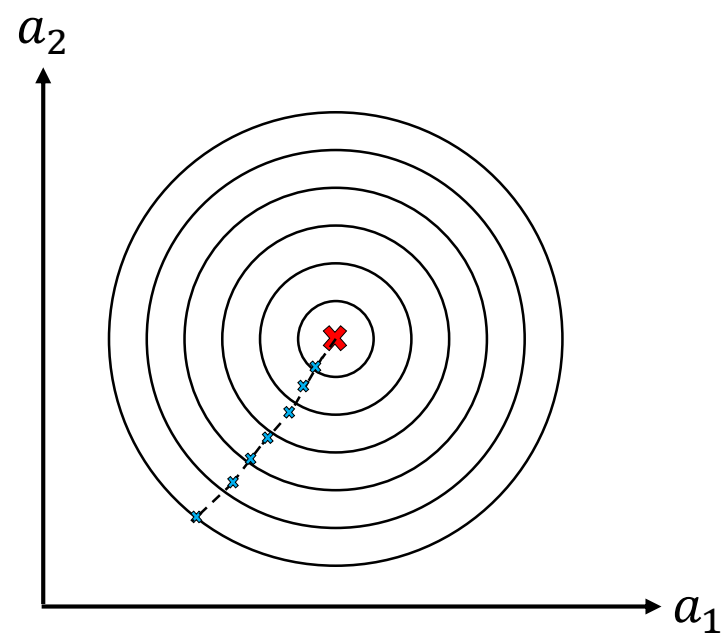
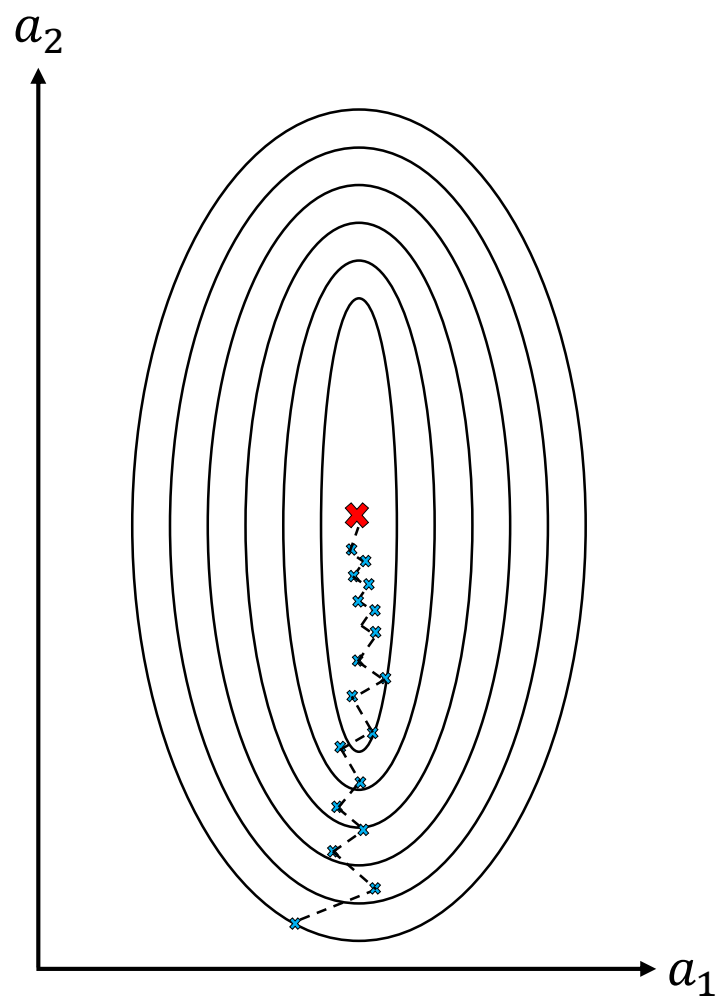


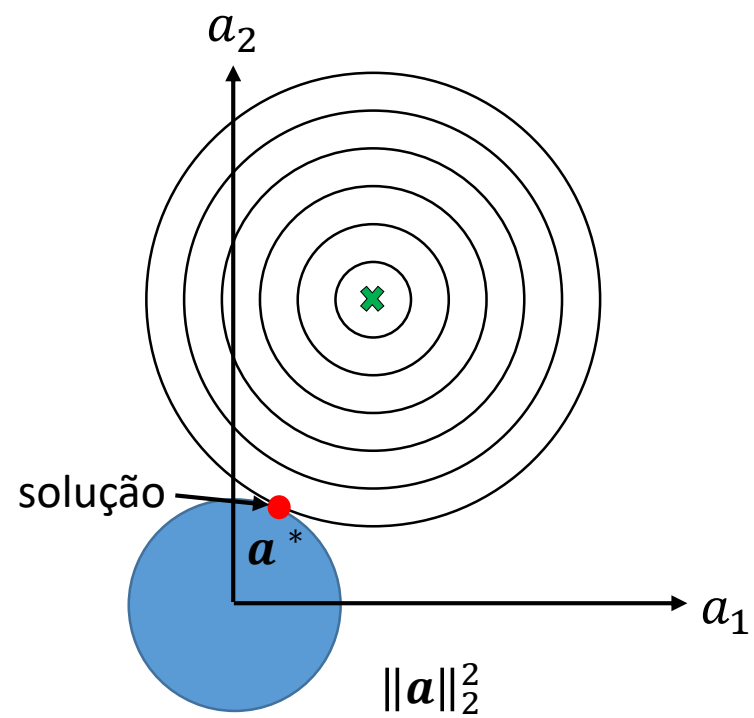
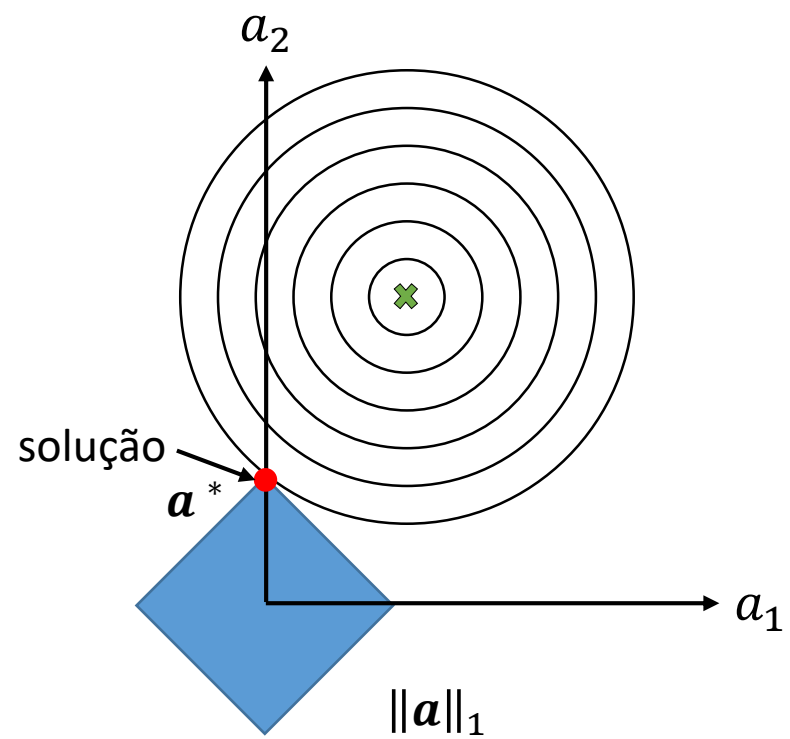
gradiente negativo: $a_1 = a_1^{\text{inicial}} + \alpha \nabla J_e(a_1)$
 a_1 aumenta e se aproxima do mínimo



gradiente positivo: $a_1 = a_1^{\text{inicial}} - \alpha \nabla J_e(a_1)$
 a_1 diminuiu e se aproxima do mínimo







Gradiente Descendente Estocástico

