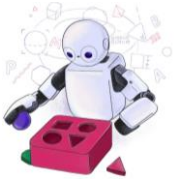
Slide 1

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



Inatel

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

Slide 2

Recapitulando

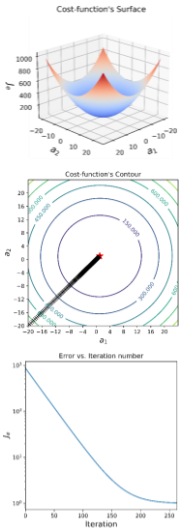
- Discutimos sobre o vetor gradiente.
- Aprendemos dois algoritmos que usam o vetor gradiente para a resolução de problemas de otimização.
- Vimos as três versões do gradiente descendente, como implementá-las em Python e as comparamos.
- Nesta parte, discutiremos o quão importante é o ajuste do passo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

Slide 3

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***.

Exemplo: linear_regression_selecting_the_learning_rate.ipynb



Exemplo:
https://mybinder.org/v2/gh/zz4fap/t319_aprendizado_de_maquina/main?filepath=notebooks%2Fregression%2Flinear_regression_selecting_the_learning_rate.ipynb

Se o passo de aprendizagem for muito pequeno, o algoritmo precisará passar por muitas iterações para convergir, o que levará muito tempo.

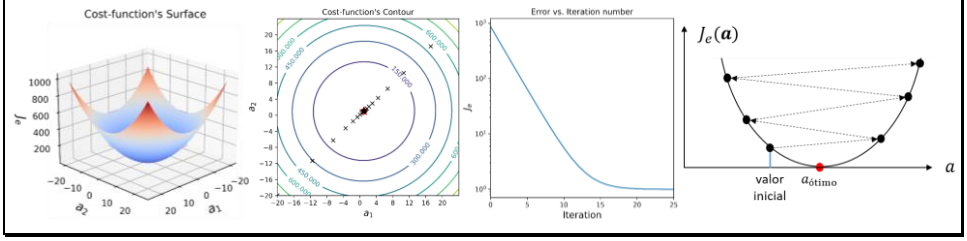
Por outro lado, se ele for muito grande, você pode pular o vale e acabar do outro lado, possivelmente até mais alto do que antes. Isso pode fazer o algoritmo divergir, com valores cada vez maiores, falhando em encontrar uma boa solução.

Assim, o passo de aprendizagem deve ser experimentado/explorado para encontrar o melhor valor que acelere a descida do gradiente.

Slide 4

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.



Enquanto o sentido e a direção são determinados pelo vetor gradiente da função de erro, a taxa de aprendizado determina o quão grande um passo é dado nessa direção.

Se o passo de aprendizagem for muito pequeno, o algoritmo precisará passar por muitas iterações para convergir, o que levará muito tempo.

Por outro lado, se ele for muito grande, você pode pular o vale e acabar do outro lado, possivelmente até mais alto do que antes. Isso pode fazer o algoritmo divergir, com valores cada vez maiores, falhando em encontrar uma boa solução.

Assim, o passo de aprendizagem deve ser experimentado/explorado para encontrar o melhor valor que acelere a descida do gradiente.

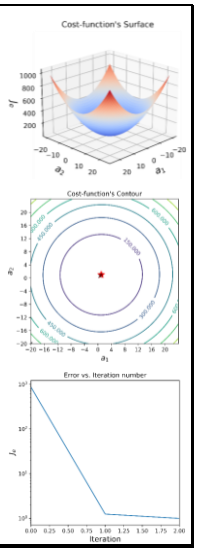
Ao usar grandes valores para o passo de aprendizagem, é possível encontrar um ciclo de feedback positivo no qual grandes valores induzem grandes gradientes que, então, induzem uma grande atualização dos pesos. Se essas atualizações aumentarem consistentemente o tamanho dos pesos, então [os pesos] se afastam rapidamente do ponto de mínimo até que ocorra o estouro da precisão numérica.

Uma boa intuição para se ter em mente é que, com uma alta taxa de aprendizado o vetor de pesos "oscila" ou "pula" caoticamente, incapaz de convergir para áreas mais profundas, porém mais estreitas, da superfície de erro.

Slide 5

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.



Se o passo de aprendizagem for muito pequeno, o algoritmo precisará passar por muitas iterações para convergir, o que levará muito tempo.

Por outro lado, se ele for muito grande, você pode pular o vale e acabar do outro lado, possivelmente até mais alto do que antes. Isso pode fazer o algoritmo divergir, com valores cada vez maiores, falhando em encontrar uma boa solução.

Assim, o passo de aprendizagem deve ser experimentado/explorado para encontrar o melhor valor que acelere a descida do gradiente.

O momentum adiciona uma fração da atualização de peso anterior a atual. Quando o gradiente continua apontando na mesma direção por atualizações consecutivas, isso aumentará o tamanho dos passos dados em direção ao mínimo. Por outro lado, quando o gradiente continua mudando de direção, o momentum suaviza as variações, ou seja, as atualizações.

OBS.: Passos largos durante as iterações iniciais e curtos conforme o algoritmo se aproxima do mínimo podem acelerar a convergência. Este tipo de abordagem é implementada por ***esquemas de variação programada*** do passo de aprendizagem

- Por exemplo: momentum, anelamento, algoritmos de otimização com ajuste adaptativo do passo de aprendizagem (RMSProp, AdaGrad, Adam, etc.).

Slide 6

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 ⇒ 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 ⇒ Passo pequeno demais: convergência lenta.
  - Figuras C e D ⇒ Passo grande demais: divergência.

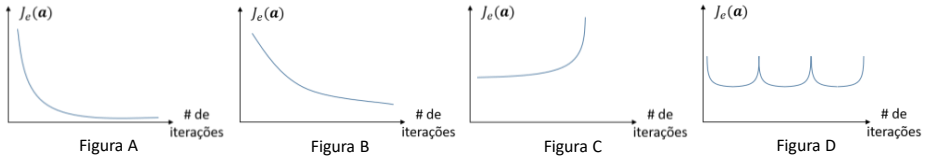
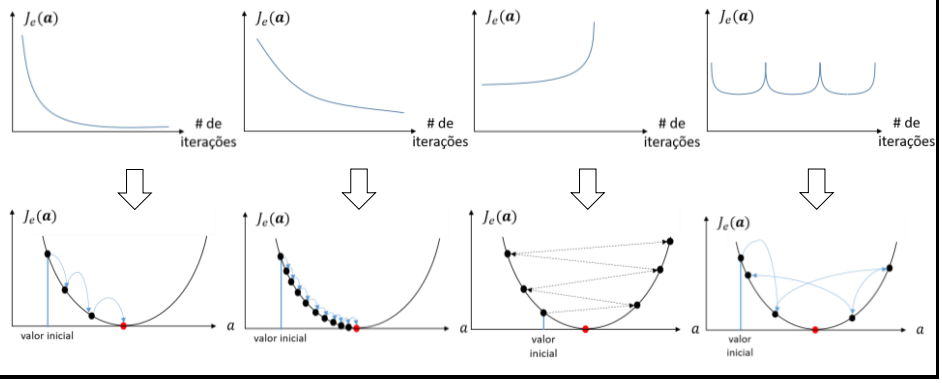


Figura A Figura B Figura C Figura D

Como você consegue debugar/depurar o algoritmo do gradiente descendente quando não é possível se plotar o gráfico de contorno e verificar o caminho seguido pelo algoritmo?

Slide 7

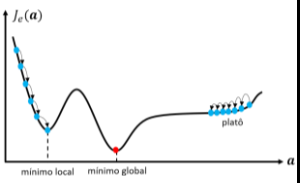
Como depurar o algoritmo do GD?



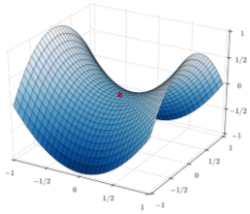
Slide 8

Desafios Encontrados pelo Gradiente Descente

- 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.



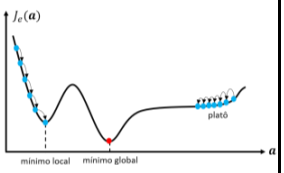
- Além disso, como vimos antes, passos grandes podem desestabilizar o algoritmo e passos muito pequenos aumentar demais o tempo de convergência.

Platô: terreno elevado e plano.

Slide 9

Desafios Encontrados pelo Gradiente Descente

- A figura mostra dois dos principais desafios encontrados pelo ***algoritmo do gradiente descente***. 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.



Platô: terreno elevado e plano.

A Figura acima mostra os dois principais desafios do Gradiente Descente: se a inicialização aleatória iniciar o algoritmo à esquerda, convergirá para um mínimo local, que não é tão bom quanto o mínimo global. Se começar à direita, levará muito tempo para atravessar o platô (gradiente próximo de zero pois a inclinação é próxima de 0 graus) e, se você parar muito cedo, nunca alcançará o mínimo global.

Outro tipo de terreno irregular é o terreno em forma de sela (de cavalo). Um ponto de sela é o ponto sobre uma superfície no qual a declividade é nula, mas não se trata de um extremo local (máximo ou mínimo). É o ponto sobre uma superfície na qual a elevação é máxima numa direção e mínima na outra direção (por exemplo, na direção perpendicular).

Dado que o passo de aprendizagem seja grande o suficiente como garantir que o mínimo encontrado é o global e não um mínimo local?
Muitas vezes o que se faz é treinar o modelo várias vezes, sempre inicializando os pesos de pontos diferentes, ou seja, aleatoriamente.

Referências:
https://www.offconvex.org/2016/03/22/saddlepoints/
https://bair.berkeley.edu/blog/2017/08/31/saddle-efficiency/

Slide 10

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.

COLAB:
https://colab.research.google.com/github/zz4fap/t319_aprendizado_de_maquina/blob/main/labs/Laboratorio4.ipynb

Slide 11

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, consequentemente, 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.

Um valor muito pequeno pode resultar em um longo processo de treinamento que pode ficar preso.
Um valor muito alto pode resultar na aprendizagem de um conjunto subótimo de pesos rápido demais ou em um processo de treinamento instável.

A maneira pela qual a taxa de aprendizado muda com o tempo (iteração/época) é chamada de cronograma/programa da taxa de aprendizado ou decaimento da taxa de aprendizado.

Alguns tipos de esquema para ajuste do passo de aprendizagem são:

- **Decaimento por etapas ou degraus**: reduz a taxa de aprendizado de algum fator a cada número pré-definido de iterações ou épocas. Os valores típicos são utilizados para reduzir a taxa de aprendizado pela metade a cada número pré-definido de épocas. Esses números dependem muito do tipo de problema e do modelo. Uma heurística que você pode ver na prática é observar o erro de validação durante o treinamento com uma taxa de aprendizado fixa e reduzir a taxa de aprendizado em uma constante (por exemplo, 0,5) sempre que o erro de validação parar de decrescer.
- **Decaimento exponencial**: tem a forma matemática $\alpha = \alpha 0\ e^{(-kt)}$, onde $\alpha 0$, k são hiperparâmetros e t é o número da iteração (mas você também pode usar o número de épocas).
- **Decaimento temporal**: tem a forma matemática $\alpha = \alpha 0 / (1 + kt)$, onde a0, k são hiperparâmetros e t é o número da iteração.

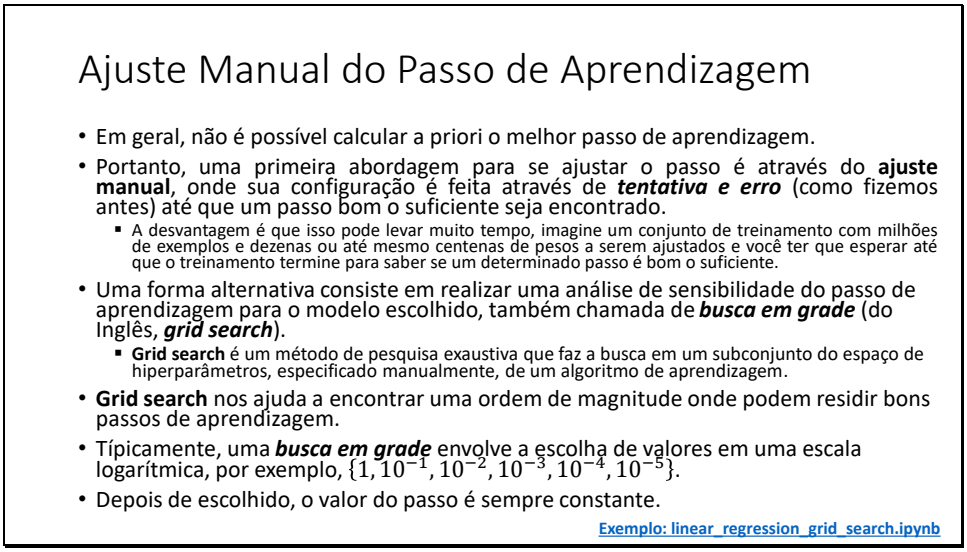
Modificação Adaptativa:
As abordagens anteriores manipulam a taxa de aprendizado global e igualmente para todos os parâmetros. Ajustar a taxa de aprendizado é um processo caro, muito

trabalho foi desenvolvido para a criação de métodos que possam ajustar adaptativamente as taxas de aprendizado, e até fazê-lo por parâmetro. Muitos desses métodos ainda podem exigir outras configurações de hiperparâmetro, mas o argumento é que eles são bem-comportados para uma faixa mais ampla de valores de hiperparâmetro do que o ajuste do passo de aprendizado.

Referências:

[1] https://machinelearningmastery.com/learning-rate-for-deep-learning-neural-networks/

Slide 12



Exemplo:
https://mybinder.org/v2/gh/zz4fap/t319_aprendizado_de_maquina/main?filepath=notebooks%2Fregression%2Flinear_regression_grid_search.ipynb

A maneira pela qual a taxa de aprendizado muda com o tempo (iteração/época) é chamada de cronograma/programa da taxa de aprendizado ou decaimento da taxa de aprendizado.

Alguns tipos de esquema para ajuste do passo de aprendizagem são:

- **Decaimento por etapas ou degraus**: reduz a taxa de aprendizado de algum fator a cada número pré-definido de iterações ou épocas. Os valores típicos são utilizados para reduzir a taxa de aprendizado pela metade a cada número pré-definido de épocas. Esses números dependem muito do tipo de problema e do modelo. Uma heurística que você pode ver na prática é observar o erro de validação durante o treinamento com uma taxa de aprendizado fixa e reduzir a taxa de aprendizado em uma constante (por exemplo, 0,5) sempre que o erro de validação parar de decrescer.
- **Decaimento exponencial**: tem a forma matemática α = α0 e^(-kt), onde α0, k são hiperparâmetros e t é o número da iteração (mas você também pode usar o número de épocas).
- **Decaimento temporal**: tem a forma matemática α = α0 / (1 + kt), onde a0, k são hiperparâmetros e t é o número da iteração.

Modificação Adaptativa:
As abordagens anteriores manipulam a taxa de aprendizado global e igualmente para todos os parâmetros. Ajustar a taxa de aprendizado é um processo caro, muito trabalho foi desenvolvido para a criação de métodos que possam ajustar

adaptativamente as taxas de aprendizado, e até fazê-lo por parâmetro. Muitos desses métodos ainda podem exigir outras configurações de hiperparâmetro, mas o argumento é que eles são bem-comportados para uma faixa mais ampla de valores de hiperparâmetro do que o ajuste do passo de aprendizado.

Slide 13

Ajuste do Passo de Aprendizagem por Redução Programada

- No treinamento com o gradiente descendente, geralmente é útil diminuir a taxa de aprendizado ao longo do tempo.
- Seria interessante que no incio do treinamento o algoritmo desse passos largos e aprendesse mais rápido e conforme ele se aproximasse do mínimo, gostaríamos que ele desse passos mais curtos para não ultrapassá-lo (muito benéfico para GDE e Mini-batch).
- Portanto, para se obter uma convergência mais rápida, evitar oscilações e ficar preso em mínimos locais, o passo de aprendizagem é geralmente variado durante o treinamento, de acordo com um ***esquema de variação (em etapas) do passo de aprendizagem***, também chamado de ***redução programada***.

A maneira pela qual a taxa de aprendizado muda com o tempo (iteração/época) é chamada de cronograma/programa da taxa de aprendizado ou decaimento da taxa de aprendizado.

Alguns tipos de esquema para ajuste do passo de aprendizagem são:

- **Decaimento por etapas ou degraus**: reduz a taxa de aprendizado de algum fator a cada número pré-definido de iterações ou épocas. Os valores típicos são utilizados para reduzir a taxa de aprendizado pela metade a cada número pré-definido de épocas. Esses números dependem muito do tipo de problema e do modelo. Uma heurística que você pode ver na prática é observar o erro de validação durante o treinamento com uma taxa de aprendizado fixa e reduzir a taxa de aprendizado em uma constante (por exemplo, 0,5) sempre que o erro de validação parar de decrescer.
- **Decaimento exponencial**: tem a forma matemática $\alpha = \alpha 0\ e^{(-kt)}$, onde $\alpha 0$, k são hiperparâmetros e t é o número da iteração (mas você também pode usar o número de épocas).
- **Decaimento temporal**: tem a forma matemática $\alpha = \alpha 0 / (1 + kt)$, onde a0, k são hiperparâmetros e t é o número da iteração.

Modificação Adaptativa:

As abordagens anteriores manipulam a taxa de aprendizado global e igualmente para todos os parâmetros. Ajustar a taxa de aprendizado é um processo caro, muito trabalho foi desenvolvido para a criação de métodos que possam ajustar adaptativamente as taxas de aprendizado, e até fazê-lo por parâmetro. Muitos desses métodos ainda podem exigir outras configurações de hiperparâmetro, mas o argumento é que eles são bem-comportados para uma faixa mais ampla de valores de hiperparâmetro do que o ajuste do passo de aprendizado.

Slide 14

Ajuste do Passo de Aprendizagem por Redução Programada

- Na **redução programada**, o passo de aprendizagem tem seu valor diminuído ao longo do tempo, ou seja, ao longo do processo de treinamento (épocas ou iterações).
- Uma desvantagem é que os hiperparâmetros desses esquemas devem ser manualmente ajustados previamente e dependem do problema e do modelo adotado. Portanto, cai-se de certa forma, novamente, no problema da tentativa e erro.
- O ajuste dos hiperparâmetros dos esquemas de redução programada é muitas vezes custoso.
- Saber quando diminuir o passo de aprendizagem também pode ser complicado:
  - Se diminuirmos um grande passo lentamente, o algoritmo pode estar desperdiçando tempo saltando caoticamente com pouca melhoria no desempenho por um longo tempo.
  - Por outro lado, se diminuirmos o passo de forma muito agressiva, o modelo irá parar de aprender muito rapidamente, pois o passo vai fazer com que a atualização diminua rapidamente, incapacitando-o de alcançar o mínimo.
- Outra desvantagem é que o mesmo passo de aprendizagem é aplicado ao ajuste de todos os pesos (assim como no ajuste manual).
- **Exemplos**: decaimento exponencial, por etapas, temporal, etc.

A maneira pela qual a taxa de aprendizado muda com o tempo (iteração/época) é chamada de cronograma/programa da taxa de aprendizado ou decaimento da taxa de aprendizado.

Alguns tipos de esquema para ajuste do passo de aprendizagem são:

- **Decaimento por etapas ou degraus**: reduz a taxa de aprendizado de algum fator a cada número pré-definido de iterações ou épocas. Os valores típicos são utilizados para reduzir a taxa de aprendizado pela metade a cada número pré-definido de épocas. Esses números dependem muito do tipo de problema e do modelo. Uma heurística que você pode ver na prática é observar o erro de validação durante o treinamento com uma taxa de aprendizado fixa e reduzir a taxa de aprendizado em uma constante (por exemplo, 0,5) sempre que o erro de validação parar de decrescer.
- **Decaimento exponencial**: tem a forma matemática $\alpha = \alpha 0\ e^{(-kt)}$, onde $\alpha 0$, k são hiperparâmetros e t é o número da iteração (mas você também pode usar o número de épocas).
- **Decaimento temporal**: tem a forma matemática $\alpha = \alpha 0 / (1 + kt)$, onde a0, k são hiperparâmetros e t é o número da iteração.

Modificação Adaptativa:

As abordagens anteriores manipulam a taxa de aprendizado global e igualmente para todos os parâmetros. Ajustar a taxa de aprendizado é um processo caro, muito trabalho foi desenvolvido para a criação de métodos que possam ajustar adaptativamente as taxas de aprendizado, e até fazê-lo por parâmetro. Muitos desses métodos ainda podem exigir outras configurações de hiperparâmetro, mas o argumento é que eles são bem-comportados para uma faixa mais ampla de valores de hiperparâmetro do que o ajuste do passo de aprendizado.

Os hiperparâmetros dependem muito do tipo de problema e do modelo. Sendo utilizado.

Slide 15

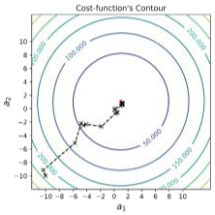
Ajuste do Passo de Aprendizagem por Redução Programada

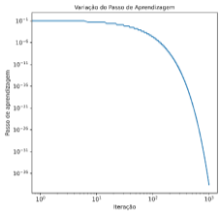
- Os três tipos mais comuns de implementação da redução programada do passo de aprendizagem são:
  - **Decaimento gradual**: também conhecido como ***decaimento por etapas*** ou ***por degraus***. Ele reduz a taxa de aprendizagem de um fator α a cada número pré-definido de iterações ou épocas, β. Um 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 são k 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.

Slide 16

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)
# Stocastic 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: stocastic_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.

Exemplo:
https://mybinder.org/v2/gh/zz4fap/t319_aprendizado_de_maquina/main?filepath=notebooks%2Fregression%2Fgd_versions%2Fstocastic_gradient_descent_with_learning_schedule_and_with_figures.ipynb

Os passos começam com grandes valores (o que ajuda a progredir rapidamente e a escapar de mínimos locais, casos em que a superfície de erro seja bastante irregular) e depois diminuem cada vez mais, permitindo que o algoritmo se estabilize no mínimo global.

Se a taxa de aprendizagem for reduzida muito rapidamente, o algoritmo poderá ficar preso no mínimo local ou até ficar travado antes de chegar ao mínimo. Se a taxa de aprendizado for reduzida muito lentamente, o algoritmo poderá oscilar ao redor do mínimo por um longo tempo e acabar com uma solução não ótima (sub-ótima) caso o treinamento se encerre muito cedo.

Alguns tipos de esquema para ajuste do passo de aprendizagem são:

- **Decaimento por etapas ou degraus**: reduz a taxa de aprendizado de algum fator a cada número pré-definido de iterações ou épocas. Os valores típicos são utilizados para reduzir a taxa de aprendizado pela metade a cada número pré-definido de épocas. Esses números dependem muito do tipo de problema e do modelo. Uma heurística que você pode ver na prática é observar o erro de validação durante o treinamento com uma taxa de aprendizado fixa e reduzir a taxa de aprendizado em uma constante (por exemplo, 0,5) sempre que o erro de validação parar de decrescer.

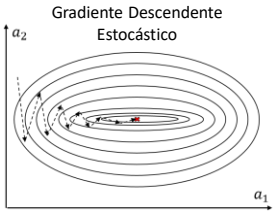
- **Decaimento exponencial**: tem a forma matemática $\alpha = \alpha 0\ e^{(-kt)}$, onde $\alpha 0$, k são hiperparâmetros e t é o número da iteração (mas você também pode usar o número de épocas).
- **Decaimento temporal**: tem a forma matemática $\alpha = \alpha 0 / (1 + kt)$, onde a0, k são hiperparâmetros e t é o número da iteração.

Exemplo: stocastic_gradient_descent_with_learning_schedule_and_with_figures.ipynb

Slide 17

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 método do momentum reduz o risco do algoritmo ficar preso em um mínimo local, bem como acelera a convergência consideravelmente em casos onde o algoritmo, de outra forma, ziguezaguearia fortemente.

O termo momentum aumenta 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.

Slide 18

Ajuste do Passo de Aprendizagem com Termo Momentum

- 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 $\boldsymbol{\nu}$ 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***.
- 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:

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

- 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*".

$$\boldsymbol{\nu}^{(i)} = -\alpha\sum_{j=0}^{i}\mu^{i-j}\frac{\partial(y\ -\hat{y})^2}{\partial \boldsymbol{a}}$$

O termo momentum aumenta 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.

Com o gradiente descendente estocástico, nós não calculamos o gradiente exato de nossa função de perda/custo/erro. Em vez disso, estamos estimando o gradiente com apenas um exemplo ou em um pequeno lote quando consideramos o mini-batch. Essa aproximação/estimação do gradiente significa que nem sempre estamos indo na direção ideal, porque as derivadas parciais são "barulhentas". Exatamente como na figura acima. Portanto, uma média exponencialmente ponderada pode nos fornecer uma estimativa melhor, que está mais próxima da derivada parcial real do que nossos cálculos ruidosos. Esta é uma das razões pelas quais o momentum pode funcionar melhor do que o SGD clássico ou o GD em Mini-batch.

O momentum pode suavizar a progressão do algoritmo de aprendizagem que, por sua vez, pode acelerar o processo de treinamento.

OBS.:
- ν é inicializado com **0** (i.e., vetor nulo).
- Os valores comuns de μ usados na prática incluem .5, .9 e .99. Assim como a taxa de aprendizado, μ também pode ser adaptado ao longo do tempo.

Referência:
https://cs231n.github.io/neural-networks-3/#sgd

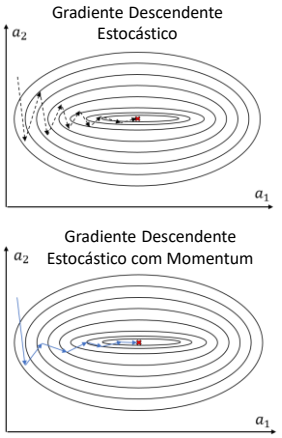
Slide 19

Ajuste do Passo de Aprendizagem com Termo Momentum

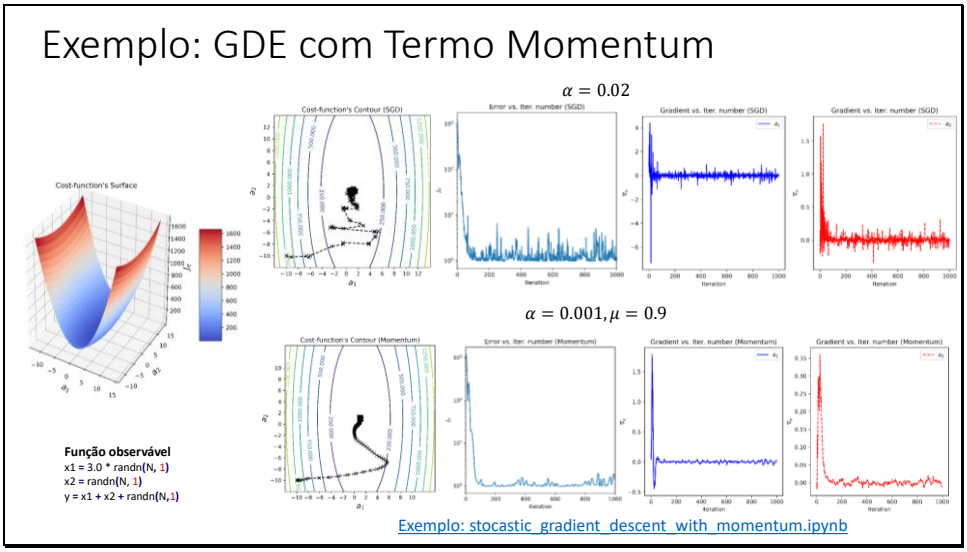
- Ao analisarmos a equação de atualização dos pesos, vemos que a variável $\boldsymbol{v}$ faz com que a atualização seja maior para dimensões cujos gradientes apontam nas mesmas direções e reduz atualizações para dimensões cujos gradientes mudam de direção. Como resultado, temos convergência mais rápida e oscilação reduzida.

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

- O efeito do algoritmo do momentum no GDE é ilustrado na figura ao lado.
- Curiosidade, o nome ***momentum*** deriva de uma analogia com a física, em que o ***gradiente negativo*** é uma força que move uma ***partícula*** através do ***espaço de parâmetros***, de acordo com as leis de movimento de Newton.
- Em física, ***momentum*** é igual a massa vezes a velocidade de uma partícula. No ***algoritmo do momentum***, assumimos massa unitária, então o vetor ***velocidade*** $\boldsymbol{v}$ também pode ser considerado como o momentum da partícula.



Slide 20



Exemplo:
https://mybinder.org/v2/gh/zz4fap/t319_aprendizado_de_maquina/main?filepath=notebooks%2Fregression%2Fgd_versions%2Fstocastic_gradient_descent_with_momentum.ipynb

Slide 21

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.

A maneira pela qual a taxa de aprendizado muda com o tempo (iteração/época) é chamada de cronograma/programa da taxa de aprendizado ou decaimento da taxa de aprendizado.

Alguns tipos de esquema para ajuste do passo de aprendizagem são:

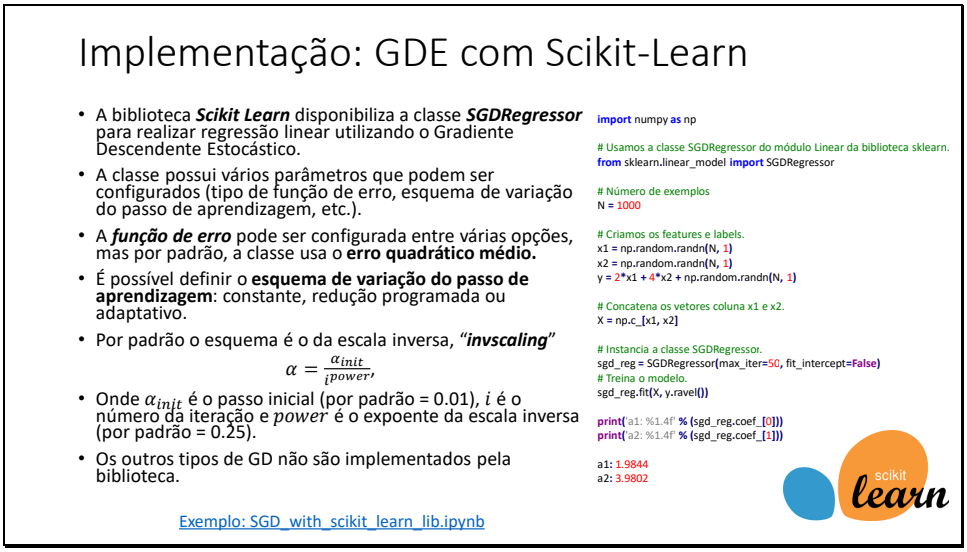
- **Decaimento por etapas ou degraus**: reduz a taxa de aprendizado de algum fator a cada número pré-definido de iterações ou épocas. Os valores típicos são utilizados para reduzir a taxa de aprendizado pela metade a cada número pré-definido de épocas. Esses números dependem muito do tipo de problema e do modelo. Uma heurística que você pode ver na prática é observar o erro de validação durante o treinamento com uma taxa de aprendizado fixa e reduzir a taxa de aprendizado em uma constante (por exemplo, 0,5) sempre que o erro de validação parar de decrescer.
- **Decaimento exponencial**: tem a forma matemática $\alpha = \alpha 0\ e^{(-kt)}$, onde $\alpha 0$, k são hiperparâmetros e t é o número da iteração (mas você também pode usar o número de épocas).
- **Decaimento temporal**: tem a forma matemática $\alpha = \alpha 0 / (1 + kt)$, onde a0, k são hiperparâmetros e t é o número da iteração.

Modificação Adaptativa:

As abordagens anteriores manipulam a taxa de aprendizado global e igualmente para todos os parâmetros. Ajustar a taxa de aprendizado é um processo caro, muito trabalho foi desenvolvido para a criação de métodos que possam ajustar adaptativamente as taxas de aprendizado, e até fazê-lo por parâmetro. Muitos desses métodos ainda podem exigir outras configurações de hiperparâmetro, mas o argumento é que eles são bem-comportados para uma faixa mais ampla de valores de hiperparâmetro do que o ajuste do passo de aprendizado.

Referências:
[1] https://machinelearningmastery.com/learning-rate-for-deep-learning-neural-networks/
[2] https://ruder.io/optimizing-gradient-descent/
[3] https://towardsdatascience.com/stochastic-gradient-descent-with-momentum-a84097641a5d

Slide 22



Exemplo:
https://mybinder.org/v2/gh/zz4fap/t319_aprendizado_de_maquina/main?filepath=notebooks%2Fregression%2Fgd_versions%2FSGD_with_scikit_learn_lib.ipynb

Para executar a regressão linear usando o SGD com o Scikit-Learn, você pode usar a classe SGDRegressor, cujo padrão é otimizar a função de custo do erro ao quadrado. O código a seguir executa 50 épocas, começando com uma taxa de aprendizado de 0,1 (eta0 = 0,1), usando o cronograma de aprendizado padrão (diferente do anterior) e não usa nenhuma regularização (penalidade = Nenhuma;

Informação retirada da documentação da classe SGDRegressor (https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDRegressor.html):

learning_rate: string, default='invscaling'
'invscaling': [default]
eta = eta0 / pow(t, power_t)

power_t: double, default=0.25
The exponent for inverse scaling learning rate.

eta0: double, default=0.01
The initial learning rate for the 'constant', 'invscaling' or 'adaptive' schedules. The default value is 0.01.

Slide 23

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.

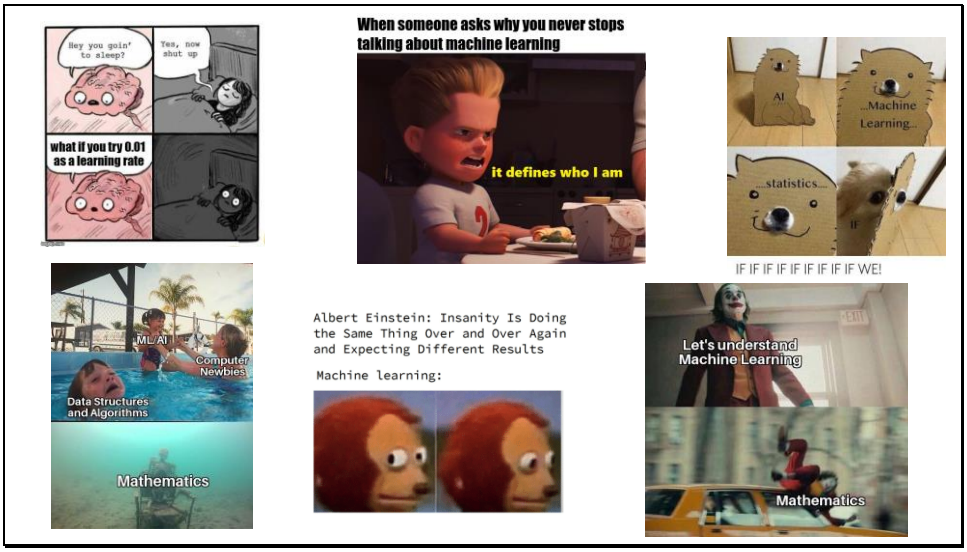
COLAB:
https://colab.research.google.com/github/zz4fap/t319_aprendizado_de_maquina/blob/main/labs/Laboratorio4.ipynb

BINDER:
https://mybinder.org/v2/gh/zz4fap/t319_aprendizado_de_maquina/main?filepath=labs%2FLaboratorio4.ipynb

Slide 24

Obrigado!

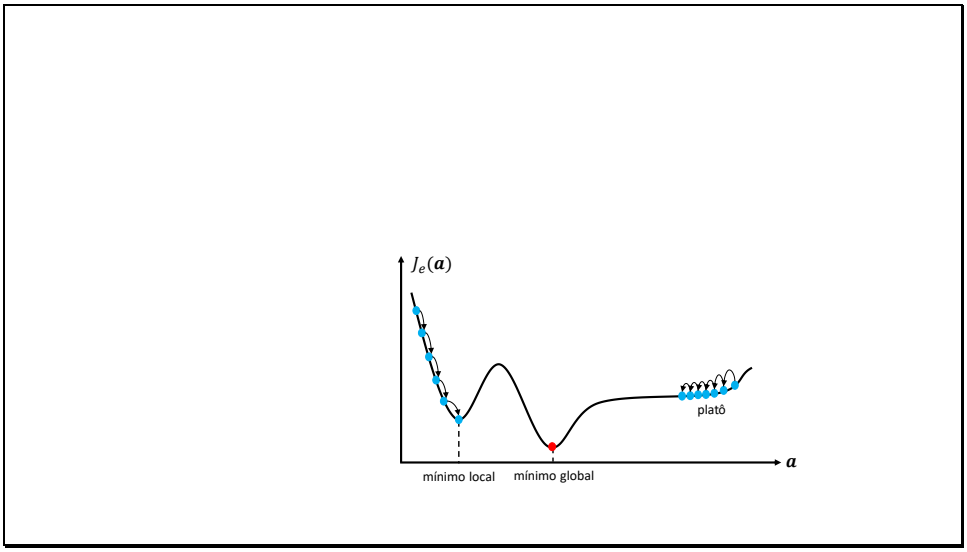
Slide 25


Hey you goin' to sleep?
Yes, now shut up
what if you try 0.01 as a learning rate
When someone asks why you never stops talking about machine learning
it defines who I am
AI
...Machine Learning...
...statistics...
IF
IF IF IF IF IF IF IF IF WE!
ML AI
Computer Newbies
Data Structures and Algorithms
Mathematics
Albert Einstein: Insanity Is Doing the Same Thing Over and Over Again and Expecting Different Results
Machine learning:
Let's understand Machine Learning
Mathematics

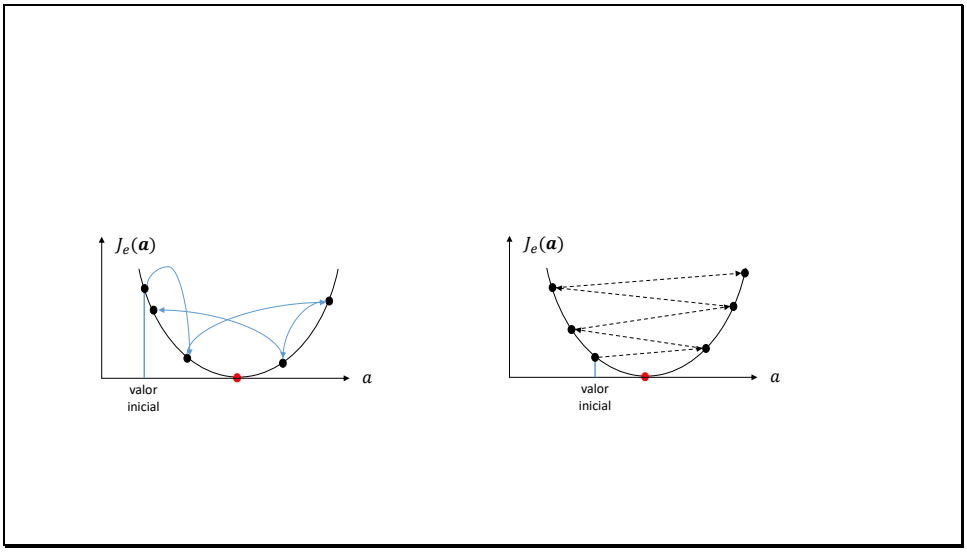
Slide 26

FIGURAS

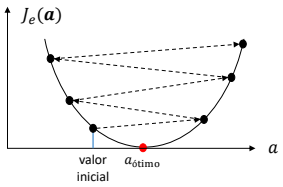
Slide 27


$J_e(\boldsymbol{a})$
platô
mínimo local
mínimo global
$\boldsymbol{a}$

Slide 28


$J_e(\boldsymbol{a})$
a
valor
inicial
$J_e(\boldsymbol{a})$
a
valor
inicial

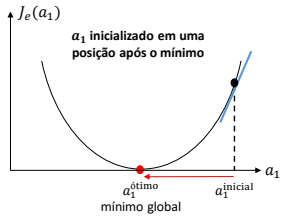
Slide 29



Slide 30

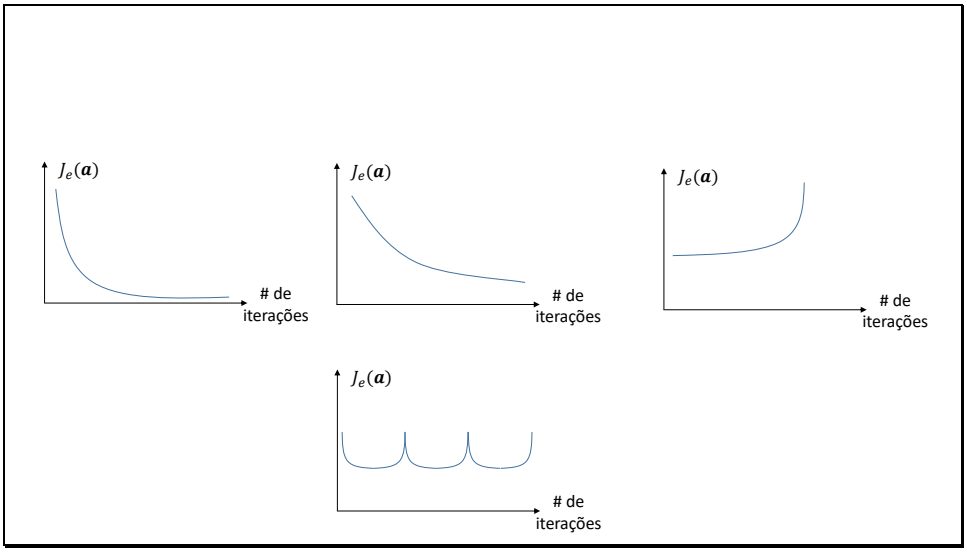
gradiente negativo: $a_1 = a_1^{\text{inicial}} + \alpha \nabla J_e(a_1)$
a_1 aumenta e se aproxima do mínimo

Slide 31

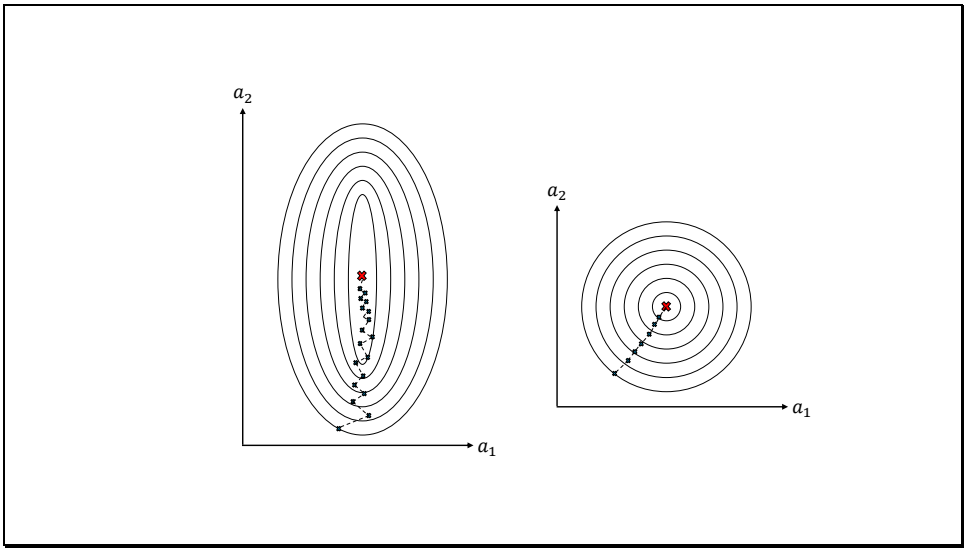


gradiente positivo: $a_1 = a_1^{\text{inicial}} - \alpha \nabla J_e(a_1)$
a_1 diminiu e se aproxima do mínimo

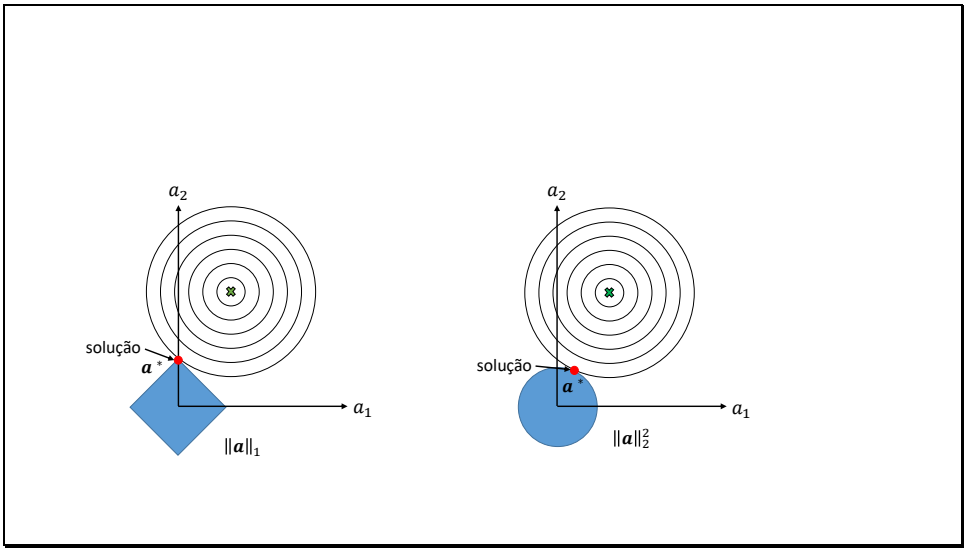
Slide 32


$J_e(\boldsymbol{a})$
de iterações
$J_e(\boldsymbol{a})$
de iterações
$J_e(\boldsymbol{a})$
de iterações
$J_e(\boldsymbol{a})$
de iterações

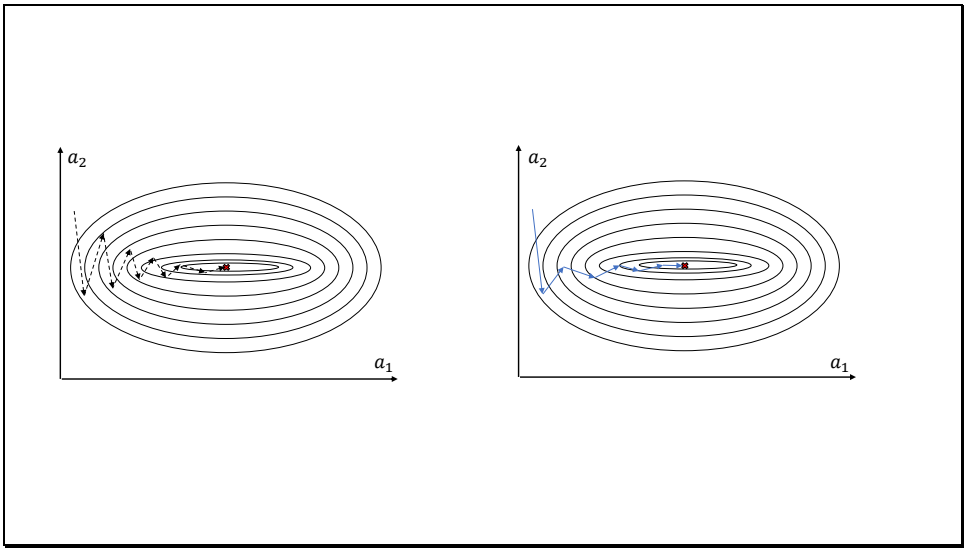
Slide 33



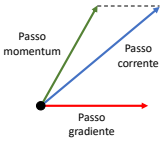
Slide 34


a_2
solução
$\boldsymbol{a}^*$
a_1
$\|\boldsymbol{a}\|_1$
a_2
solução
$\boldsymbol{a}^*$
a_1
$\|\boldsymbol{a}\|_2^2$

Slide 35



Slide 36



O termo momentum aumenta 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.