Dự án Cuối kỳ

Môn: Nhập môn Học Máy

**Bài 1 (3 điểm): làm riêng từng người**

Trình bày một bài nghiên cứu, đánh giá của em về các vấn đề sau:

1. Tìm hiểu, so sánh các phương pháp Optimizer trong huấn luyện mô hình học máy;
2. Tìm hiểu về Continual Learning và Test Production khi xây dựng một giải pháp học máy để giải quyết một bài toán nào đó.

Nộp file .doc hoặc .pdf.

Trình bày trên cả Github

MSSV: 521H0324

Họ Tên: Nguyễn Văn Trường

MỤC LỤC

[I. Tìm hiểu, so sánh các phương pháp Optimizer trong huấn luyện mô hình học máy; 1](#_Toc154093033)

[1. Gradient Descent (GD): 1](#_Toc154093034)

[a. Gradient cho hàm 1 biến : 2](#_Toc154093035)

[b. Gradient descent cho hàm nhiều biến : 5](#_Toc154093036)

[2. Stochastic Gradient Descent (SGD): 8](#_Toc154093037)

[a. Giảm dần learning rate: 9](#_Toc154093038)

[3. Gradient Descent với Momentum: 13](#_Toc154093039)

[4. Adagrad: 18](#_Toc154093040)

[5. RMSprop: 21](#_Toc154093041)

[6. Adam: 22](#_Toc154093042)

[II. Tìm hiểu về Continual Learning và Test Production khi xây dựng một giải pháp học máy để giải quyết một bài toán nào đó. 24](#_Toc154093043)

[1. Continual Learning. 24](#_Toc154093044)

[a. Continual Learning là gì? 24](#_Toc154093045)

[b. Các loại học máy liên tục: 26](#_Toc154093046)

[c. Quá trình học tập liên tục: 26](#_Toc154093047)

[d. Ưu điểm của việc học liên tục: 27](#_Toc154093048)

[e. Nhược điểm của việc học tập liên tục: 28](#_Toc154093049)

[f. Học tập liên tục cho trí tuệ nhân tạo và học máy: 28](#_Toc154093050)

[g. Ứng dụng của việc học liên tục: 29](#_Toc154093051)

[2. Test Production: 29](#_Toc154093052)

[a. Test Production là gì ? 30](#_Toc154093053)

[b. Lợi ích của Test Production: 30](#_Toc154093054)

[c. Tại sao cần Test Production? 31](#_Toc154093055)

[d. Một số mẹo khi Test Production: 31](#_Toc154093056)

[e. Kết luận : 32](#_Toc154093057)

[Tài Liệu Tham Khảo 33](#_Toc154093058)

**Bài Làm**

1. Tìm hiểu, so sánh các phương pháp Optimizer trong huấn luyện mô hình học máy;

* Optimizer là một phần quan trọng của quá trình huấn luyện mô hình học máy. Khi huấn luyện một mô hình, chúng ta thường định nghĩa một hàm mất mát (loss function) để đo lường sự khác biệt giữa dự đoán của mô hình và giá trị thực tế. Mục tiêu của chúng ta là cực tiểu hóa hàm mất mát này.
* Optimizer là công cụ quyết định cách mà trọng số của mô hình được điều chỉnh để giảm thiểu hàm mất mát này. Nó là người điều chỉnh các tham số của mô hình dựa trên gradient của hàm mất mát, để mô hình học từ dữ liệu và cải thiện dự đoán của nó.

Các phương pháp optimizer :

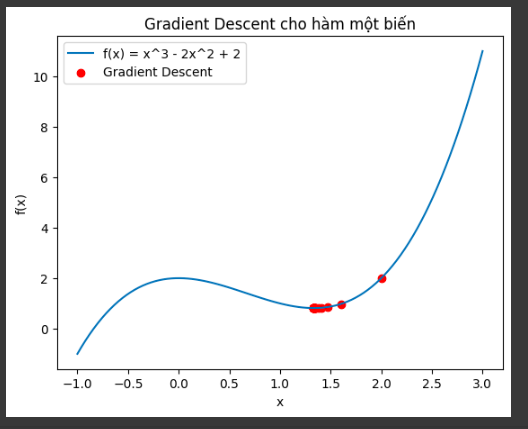
1. Gradient Descent (GD):

* Trong Machine Learning nói riêng và Toán Tối ưu nói chung, chúng ta thường xuyên phải tìm giá trị nhỏ nhất (hoặc đôi khi là lớn nhất) của một số hàm nào đó.
* Ví dụ như các hàm mất mát trong hai bài Hồi quy tuyến tính và Phân cụm K-mean. Nhìn chung, việc tìm kiếm mức tối thiểu toàn cầu của các chức năng bị mất mát trong Machine Learning là rất phức tạp, thậm chí là bất khả thi. Thay vào đó, người ta thường cố gắng tìm kiếm các điểm tối thiểu cục bộ và ở một mức độ nào đó, coi đó là thử nghiệm cần tìm của bài toán.
* Điểm tối thiểu cục bộ là thử nghiệm của phương pháp đạo giáo hàm bằng 0. Nếu bằng một cách nào đó có thể tìm được toàn bộ (hữu hạn) các điểm cực tiểu, ta chỉ cần thay từng điểm tối thiểu cục bộ vào hàm số rồi tìm điểm làm cho hàm có giá trị nhỏ nhất (đoạn này nghe rất quen thuộc, đúng không?). Tuy nhiên, trong hầu hết các trường hợp, việc giải phương trình đạo hàm bằng 0 là bất khả thi. Nguyên nhân có thể đến từ các dạng phức tạp của hàm đạo đức, từ các dữ liệu điểm có số chiều lớn hoặc từ các công việc có quá nhiều dữ liệu điểm.
* Hướng tiếp cận phổ biến nhất là xuất phát từ một điểm mà chúng ta coi là gần với thử nghiệm của bài toán, sau đó sử dụng một phép toán để tiến dần đến điểm cần tìm, tức đến khi đạo hàm gần với 0. gradient Descent (Viết gọn là GD) và các biến thể của nó là một trong những phương pháp được sử dụng nhiều nhất.

**Công thức : xnew = xold - learningrate.gradient(x)**

1. Gradient cho hàm 1 biến :

* import numpy as np
* import matplotlib.pyplot as plt
* # Hàm số
* def f(x):
* return x\*\*3 - 2\*x\*\*2 + 2
* # Đạo hàm của hàm số
* def df(x):
* return 3\*x\*\*2 - 4\*x
* # Gradient Descent cho hàm một biến
* def gradient\_descent\_1d(x\_start, learning\_rate, epochs):
* x\_values = [x\_start]
* for i in range(epochs):
* gradient = df(x\_values[-1])
* x\_new = x\_values[-1] - learning\_rate \* gradient
* x\_values.append(x\_new)
* return x\_values
* # Thiết lập và chạy Gradient Descent
* x\_start = 2 # Giá trị ban đầu của x
* learning\_rate = 0.1
* epochs = 20
* x\_values = gradient\_descent\_1d(x\_start, learning\_rate, epochs)
* # Vẽ biểu đồ
* x = np.linspace(-1, 3, 100)
* plt.plot(x, f(x), label='f(x) = x^3 - 2x^2 + 2')
* plt.scatter(x\_values, f(np.array(x\_values)), color='red', label='Gradient Descent')
* plt.xlabel('x')
* plt.ylabel('f(x)')
* plt.legend()
* plt.title('Gradient Descent cho hàm một biến')
* plt.show()



* Gradient descent phụ thuộc vào nhiều yếu tố : như nếu chọn điểm x ban đầu khác nhau sẽ ảnh hưởng đến quá trình hội tụ; hoặc tốc độ học (learning rate) quá lớn hoặc quá nhỏ cũng ảnh hưởng: nếu tốc độ học quá nhỏ thì tốc độ hội tụ rất chậm ảnh hưởng đến quá trình training, còn tốc độ học quá lớn thì tiến nhanh tới đích sau vài vòng lặp tuy nhiên thuật toán không hội tụ, quanh quẩn quanh đích vì bước nhảy quá lớn.

1. Gradient descent cho hàm nhiều biến :

import numpy as np

import matplotlib.pyplot as plt

# Hàm số

def f(x, y):

return x\*\*2 + y\*\*2

# Gradient của hàm số

def gradient(x, y):

return np.array([2\*x, 2\*y])

# Gradient Descent cho hàm nhiều biến

def gradient\_descent\_2d(x\_start, y\_start, learning\_rate, epochs):

x\_values = [x\_start]

y\_values = [y\_start]

for i in range(epochs):

grad = gradient(x\_values[-1], y\_values[-1])

x\_new = x\_values[-1] - learning\_rate \* grad[0]

y\_new = y\_values[-1] - learning\_rate \* grad[1]

x\_values.append(x\_new)

y\_values.append(y\_new)

return x\_values, y\_values

# Thiết lập và chạy Gradient Descent

x\_start, y\_start = 3, 4 # Giá trị ban đầu của x và y

learning\_rate = 0.1

epochs = 20

x\_values, y\_values = gradient\_descent\_2d(x\_start, y\_start, learning\_rate, epochs)

# Vẽ đồ thị

fig = plt.figure()

ax = fig.add\_subplot(111, projection='3d')

x = np.linspace(-5, 5, 100)

y = np.linspace(-5, 5, 100)

x, y = np.meshgrid(x, y)

z = f(x, y)

ax.plot\_surface(x, y, z, alpha=0.5)

ax.scatter(x\_values, y\_values, f(np.array(x\_values), np.array(y\_values)), color='red')

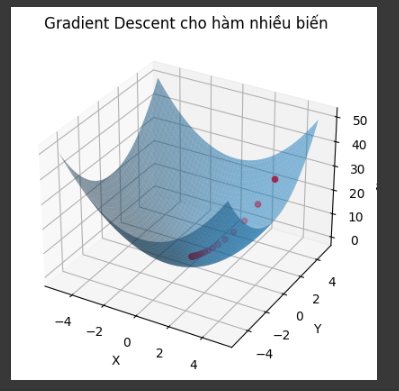
ax.set\_xlabel('X')

ax.set\_ylabel('Y')

ax.set\_zlabel('f(X, Y)')

ax.set\_title('Gradient Descent cho hàm nhiều biến')

plt.show()



Ưu điểm:

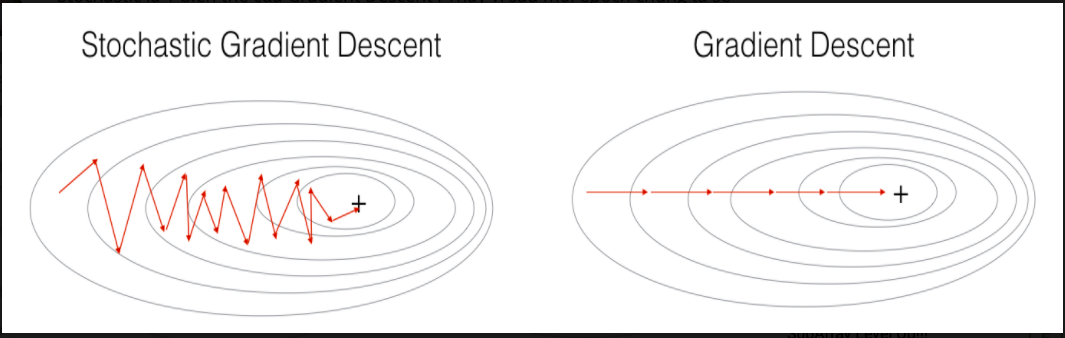
* Đơn giản và hiệu quả: Gradient descent là một phương pháp đơn giản và dễ triển khai. Nó cực kỳ hiệu quả trong việc tối ưu hóa các hàm mục tiêu đa chiều.
* Khả năng tìm điểm cực tiểu địa phương: Gradient descent có thể tìm điểm cực tiểu địa phương của hàm mục tiêu, giúp thuật toán hội tụ đến điểm tối ưu.
* Áp dụng rộng rãi: Thuật toán này có thể được áp dụng cho nhiều mô hình machine learning khác nhau và trong nhiều lĩnh vực khác nhau.

Nhược điểm:

* Dễ rơi vào điểm cực tiểu địa phương: Gradient descent có thể dễ dàng rơi vào điểm cực tiểu địa phương thay vì điểm cực tiểu toàn cục, đặc biệt là khi hàm mục tiêu không phải lúc nào cũng là một hàm lồi.
* Phụ thuộc vào learning rate: Lựa chọn learning rate (tốc độ học) là một yếu tố quan trọng. Nếu chọn learning rate quá lớn, thuật toán có thể không hội tụ hoặc dao động quanh điểm cực tiểu. Ngược lại, nếu chọn learning rate quá nhỏ, thuật toán có thể hội tụ rất chậm.
* Cần xử lý đặc trưng đầu vào: Khi sử dụng gradient descent trong machine learning, việc xử lý và chuẩn hóa dữ liệu đầu vào thường là cần thiết để giúp thuật toán hội tụ nhanh hơn và tránh các vấn đề như biến động của dữ liệu.
* Khó xử lý với các hàm không liên tục hoặc không khả vi: Gradient descent yêu cầu hàm mục tiêu là liên tục và khả vi tại mọi điểm trong không gian của nó, điều này có thể là một hạn chế đối với các bài toán có hàm mục tiêu phức tạp.

1. Stochastic Gradient Descent (SGD):

* Thay vì sử dụng toàn bộ tập huấn luyện thì Stochastics Gradient Descent (SGD) sẽ lấy ngẫu nhiên 1 phần tử ở tập huấn luyện và thực hiện tính lại vector độ dốc dựa chỉ dựa trên 1 điểm dữ liệu, sau đó lặp đi lặp lại đến khi kết thúc. Và việc tính toán dựa trên 1 điểm dữ liệu sẽ khiến thuật toán chạy nhanh hơn bởi có rất ít dữ liệu cần xử lý ở mỗi vòng lặp. Và điều này cũng giúp mô hình có thể được huấn luyện với những dữ liệu lớn vì mỗi vòng lặp chỉ cần đưa 1 điểm dữ liệu vào trong bộ nhớ.
* Mặt khác do tính chất ngẫu nhiên của dữ liệu đưa vào nên trong quá trình huấn luyện, thay vì hàm chi phí giảm từ từ giống Batch GD thì hàm chi phí của SGD sẽ lúc tăng lúc giảm nhưng sẽ giảm dần theo khoảng thời gian. Dần dần nghiệm của bài toán nó sẽ tiệm cận rất gần với cực tiểu nhưng khi đã đạt được cực tiểu thì giá trị hàm chi phí sẽ liên tục thay đổi mà không giữ ổn định. Khi gặp điều kiện dừng ta sẽ được bộ tham số cuối cùng đủ tốt, nhưng chưa thật sự tối ưu.
* Khi hàm chi phí liên tục thay đổi có thể giúp thuật toán nhảy ra khỏi cực tiểu địa phương. Vì vậy SGD có cơ hội để tìm được cực trị toàn cục hơn là Batch Gradient Descent. Vì vậy lựa chọn ngẫu nhiên dữ liệu sẽ giúp thoát khỏi nghiệm tối ưu cục bộ nhưng điều đó nghĩa là thuật toán cũng không bao giờ có nghiệm cực tiểu.



🡺 SGD được ưa chuộng vì nó có khả năng tối ưu hóa hiệu quả trên dữ liệu lớn. Trong khi GD tính toán đạo hàm trên toàn bộ dữ liệu, SGD chỉ sử dụng một điểm dữ liệu (hoặc một vài điểm nhỏ) trong mỗi lần cập nhật, giúp giảm thiểu thời gian tính toán.

🡺 Điều này rất hữu ích khi dữ liệu thay đổi liên tục (online learning) vì SGD chỉ cần cập nhật trên dữ liệu mới mà không cần tính toán lại trên toàn bộ tập dữ liệu. Mặc dù SGD có đường đi zig-zag, nó vẫn có thể đạt được nghiệm tốt nhanh chóng, đặc biệt là trên tập dữ liệu lớn, trong khi GD có thể cần nhiều vòng lặp hơn để đạt được kết quả tương tự.

1. Giảm dần learning rate:

* Người ta nghĩ ra thêm được 1 cách giải quyết đó là giảm dần learning rate trong quá trình huấn luyện. Các vòng lặp đầu tiên chúng ta sẽ để learning rate lớn để thoát khỏi cực tiểu địa phương, sau đó giảm dần tốc độ học để đạt được cực tiểu toàn cục. Hàm để xác định learning rate ở mỗi lần lặp được gọi là hàm lên lịch cho tốc độ học.
* Nếu learning rate giảm quá nhanh, thuật toán có thể bị dừng ở điểm cực tiểu địa phương, hoặc kết thúc ở lưng chừng khi chưa đến điểm cực tiểu. Nếu learning rate giảm quá chậm, hàm chi phí có thể thay đổi lên xuống mức tối thiểu trong một thời gian dài và kết thúc bằng một nghiệm tối ưu nếu bạn dừng việc training quá sớm.

Đoạn code mô phỏng việc giảm dần learning rate trong quá trình training:

import numpy as np

import matplotlib.pyplot as plt

def loss\_function(x):

return (x - 4) \*\* 2

def gradient(x):

return 2 \* (x - 4)

def sgd\_with\_decay(initial\_lr, decay\_factor, epochs):

x = 0 # Giả sử x là tham số cần tối ưu

lr = initial\_lr

x\_values = []

losses = []

for epoch in range(epochs):

grad = gradient(x)

x = x - lr \* grad # Cập nhật tham số

lr = lr \* decay\_factor # Giảm dần learning rate

loss = loss\_function(x)

x\_values.append(x)

losses.append(loss)

print(f"Epoch {epoch + 1}/{epochs}, x = {x:.4f},

loss = {loss:.4f}, learning rate = {lr:.6f}")

return x\_values, losses

# Thiết lập tham số

initial\_learning\_rate = 0.1

decay\_factor = 0.9

num\_epochs = 50

# Chạy quá trình huấn luyện

x\_values, losses = sgd\_with\_decay(initial\_learning\_rate, decay\_factor, num\_epochs)

# Vẽ đồ thị quá trình học

plt.plot(range(1, num\_epochs + 1), losses, label='Loss')

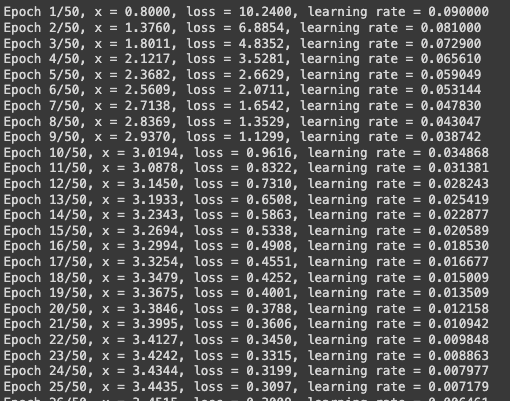
plt.xlabel('Epochs')

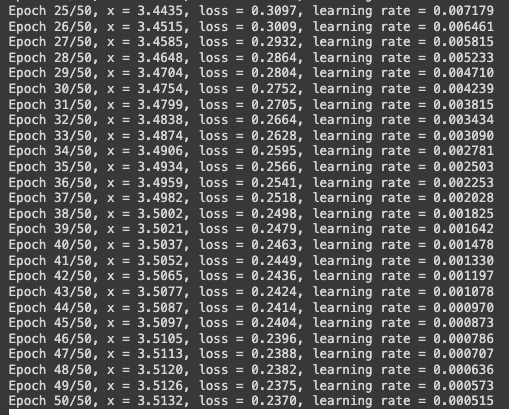
plt.ylabel('Loss')

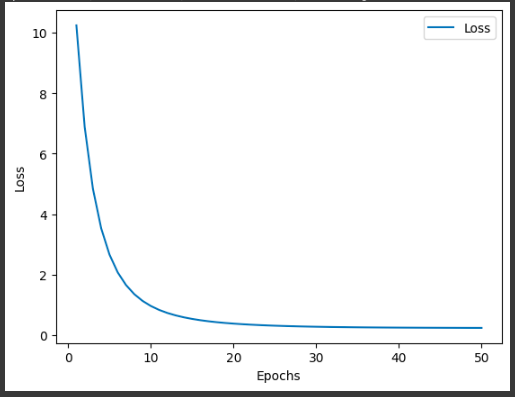
plt.legend()

plt.show()

Output:



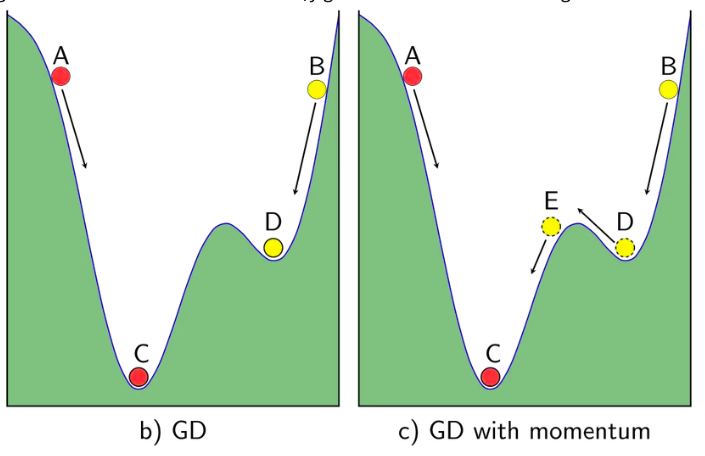




🡺 Tối ưu hóa Stochastic Gradient Descent (SGD) được ưa chuộng vì khả năng xử lý cơ sở dữ liệu lớn, điều mà Gradient Descent (GD) không thể làm. Tuy nhiên, SGD vẫn đối mặt với hai hạn chế lớn của gradient descent là cần thiết lập learning rate và điểm dữ liệu ban đầu. Để khắc phục điều này, chúng ta thường kết hợp SGD với các thuật toán khác như Momentum, AdaGrad,.. Các phương pháp này sẽ được giới thiệu chi tiết ở phần sau.

1. Gradient Descent với Momentum:

* Để khắc phục các hạn chế trên của thuật toán Gradient Descent người ta dùng gradient descent with momentum.



* Để giải thích Gradient Descent với Momentum, hãy nhìn từ góc độ vật lí: Nếu ta thả hai viên bi từ hai điểm khác nhau A và B trên một dốc, viên bi tại điểm A sẽ trượt xuống và dừng ở điểm C, trong khi viên bi ở điểm B sẽ trượt xuống và dừng ở điểm D. Tuy nhiên, nếu chúng ta muốn viên bi tại điểm B không dừng lại ở điểm D (điểm cực tiểu cục bộ) mà tiếp tục lăn tới điểm C (điểm cực tiểu toàn cục), chúng ta cần cung cấp cho viên bi đó một vận tốc ban đầu đủ lớn để vượt qua điểm E và tiếp tục đến điểm C.
* Dựa trên ý tưởng này, thuật toán Momentum được xây dựng để cung cấp đà (momentum) cho quá trình tối ưu hóa. Bằng cách tích lũy đà từ các gradient trước đó, thuật toán giúp cho việc di chuyển trên bề mặt hàm mất mát trở nên mượt mà hơn và có khả năng vượt qua các thung lũng và các điểm cực tiểu cục bộ để đạt tới điểm cực tiểu toàn cục.
* Gradient Descent với Momentum là một biến thể cải tiến của thuật toán Gradient Descent (GD) thông thường nhằm cải thiện tốc độ hội tụ và tránh được những dao động không cần thiết khi tối ưu hóa.
* Trong thuật toán GD cơ bản, việc cập nhật tham số được thực hiện dựa trên đạo hàm của hàm mất mát tại điểm hiện tại. Tuy nhiên, GD có thể di chuyển chậm hoặc bị lạc vào các điểm địa phương tối ưu. Momentum giải quyết vấn đề này bằng cách tích lũy trọng số cho các bước trước đó để tăng đà (momentum) cho việc cập nhật tham số.
* Công thức cập nhật trong Gradient Descent với Momentum là:





Trong đó:

+ θ là tham số cần tối ưu.

+ ∇J(θ) là đạo hàm của hàm mất mát theo θ.

+ η là learning rate.

+ γ là hệ số momentum, thường nằm trong khoảng (0, 1).

+ Vt là​ véc-tơ momentum tại bước thời điểm t. Nó tích lũy độ lớn và hướng của các gradient trước đó. Khi gradient theo một hướng nhất định giữ không đổi hoặc thay đổi nhẹ, momentum giúp tăng tốc độ cho quá trình di chuyển và tránh bị rơi vào các điểm cục bộ tối ưu.

* Một ví dụ nhỏ:
* import matplotlib.pyplot as plt
* # Các siêu tham số của thuật toán tối ưu
* alpha = 0.01 # Tốc độ học
* beta = 0.9 # Hệ số momentum
* # Hàm mục tiêu
* def obj\_func(x):
* return x \* x - 4 \* x + 4
* # Gradient của hàm mục tiêu
* def grad(x):
* return 2 \* x - 4
* # Tham số của hàm mục tiêu
* x = 0
* # Số lượng vòng lặp và danh sách lưu trữ giá trị của x qua các vòng lặp
* iterations = 0
* x\_values = []
* v = 0

while True:

iterations += 1

v = beta \* v + (1 - beta) \* grad(x)

x\_prev = x

x = x - alpha \* v

x\_values.append(x)

if x\_prev == x:

print("Hoàn thành quá trình tối ưu hóa.")

break

# Vẽ biểu đồ

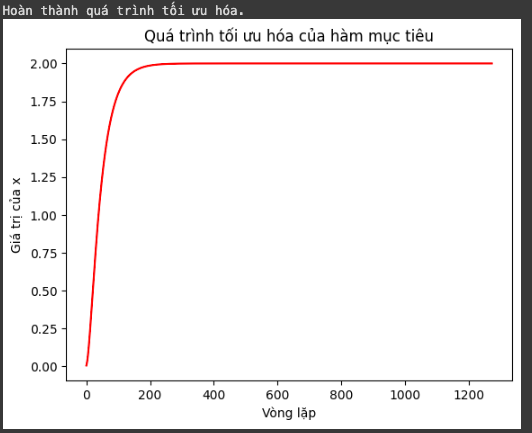
plt.plot(range(iterations), x\_values, 'r')

plt.title("Quá trình tối ưu hóa của hàm mục tiêu")

plt.xlabel("Vòng lặp")

plt.ylabel("Giá trị của x")

plt.show()



Ưu điểm:

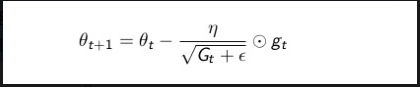
* Tăng tốc độ hội tụ: Momentum giúp tăng đà cho quá trình di chuyển của thuật toán, đặc biệt là khi đối mặt với thung lũng hoặc các điểm cực tiểu cục bộ. Điều này giúp thuật toán vượt qua các điểm đó và tiến gần hơn đến điểm tối ưu.
* Giảm dao động không cần thiết: Hệ số momentum giúp làm mịn quá trình di chuyển và giảm thiểu các dao động không mong muốn, đặc biệt khi đối mặt với nhiễu trong dữ liệu.
* Hội tụ nhanh hơn: Do đà được tích lũy qua các bước trước đó, Gradient Descent với Momentum thường hội tụ nhanh hơn so với Gradient Descent cơ bản, đặc biệt trên bề mặt hàm mất mát phức tạp.

Nhược điểm:

* Điều chỉnh hệ số momentum: Việc lựa chọn hệ số momentum có thể ảnh hưởng đến hiệu suất của thuật toán. Nếu chọn quá lớn, có thể dẫn đến việc thuật toán bị vượt quá mục tiêu hoặc dao động không ổn định.
* Khó khăn trong việc tinh chỉnh hyperparameters: Ngoài hệ số momentum, Gradient Descent với Momentum vẫn cần phải điều chỉnh learning rate (tốc độ học) và các siêu tham số khác, việc điều chỉnh các siêu tham số này có thể phức tạp.
* Khả năng vượt quá mục tiêu: Đôi khi, do đà (momentum) được tích lũy qua các bước, thuật toán có thể vượt quá mục tiêu tối ưu và không dừng lại ở điểm mong muốn.

1. Adagrad:

* Adagrad là một thuật toán tối ưu hóa trong Machine Learning, nó điều chỉnh tỷ lệ học (learning rate) của mỗi tham số dựa trên lịch sử của gradient của tham số đó. Thuật toán này được thiết kế để tự động điều chỉnh learning rate để mỗi tham số có thể được cập nhật với tỷ lệ khác nhau tùy theo mức độ thay đổi của gradient.
* Ý tưởng chính của Adagrad là sử dụng một bảng tính toán tổng bình phương của gradient đã tính toán trước đó cho mỗi tham số. Dựa trên các giá trị này, learning rate sẽ được điều chỉnh để giảm nhỏ nếu gradient lớn và tăng lên nếu gradient nhỏ, giúp tối ưu hóa tỷ lệ học cho mỗi tham số.
* Công thức cập nhật của Adagrad cho mỗi tham số w tại thời điểm t:



Trong đó:

+ n : hằng số

+ gt : gradient tại thời điểm t.

+ ϵ  : hệ số tránh lỗi ( chia cho mẫu bằng 0).

+ G : là ma trận chéo mà mỗi phần tử trên đường chéo (i,i) là bình phương của đạo hàm vectơ tham số tại thời điểm t.

* Ví dụ mô phỏng đơn giản của Adagrad trên hàm mất mát ***f*(*x*)=** với gradient là **f’(x) = 2x**

import numpy as np

import matplotlib.pyplot as plt

def loss\_function(x):

return x \*\* 2

def gradient(x):

return 2 \* x

x = 10

learning\_rate = 1

num\_iterations = 100

x\_values = []

learning\_rates = []

# Khởi tạo ma trận G với kiểu dữ liệu float64

G = np.zeros\_like(x, dtype=np.float64)

for i in range(num\_iterations):

grad = gradient(x)

G += grad \*\* 2

adjusted\_learning\_rate = learning\_rate / (np.sqrt(G) + 1e-8)

x -= adjusted\_learning\_rate \* grad

x\_values.append(x)

learning\_rates.append(adjusted\_learning\_rate)

plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)

plt.plot(x\_values, 'r')

plt.title('Giá trị của x qua các vòng lặp')

plt.xlabel('Vòng lặp')

plt.ylabel('Giá trị của x')

plt.subplot(1, 2, 2)

plt.plot(learning\_rates, 'b')

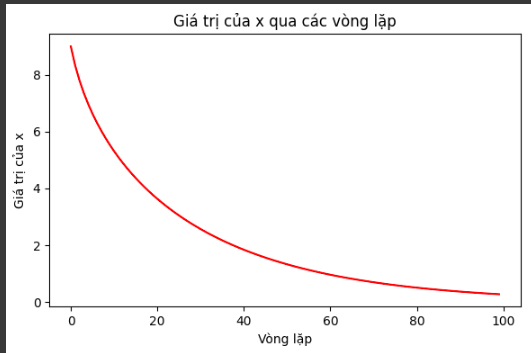
plt.title('Tỷ lệ học điều chỉnh theo thời gian')

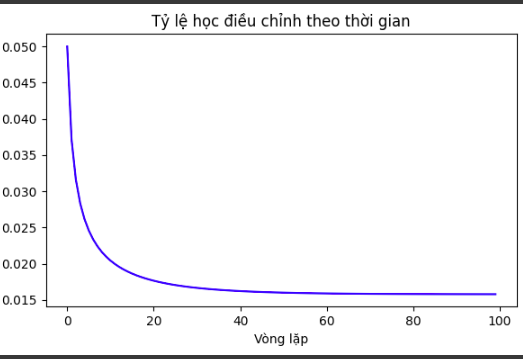
plt.xlabel('Vòng lặp')

plt.ylabel('Tỷ lệ học điều chỉnh')

plt.tight\_layout()

plt.show()





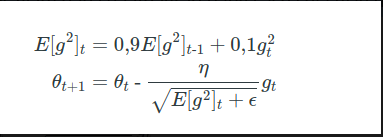
Ưu điểm :

* Một lơi ích dễ thấy của Adagrad là tránh việc điều chỉnh learning rate bằng tay, chỉ cần để tốc độ học default là 0.01 thì thuật toán sẽ tự động điều chỉnh.

Nhược điểm :

* Yếu điểm của Adagrad là tổng bình phương biến thiên sẽ lớn dần theo thời gian cho đến khi nó làm tốc độ học cực kì nhỏ, làm việc training trở nên đóng băng.

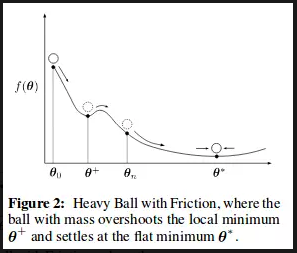
1. RMSprop:

* RMSprop giải quyết vấn đề tỷ lệ học giảm dần của Adagrad bằng cách chia tỷ lệ học cho trung bình của bình phương gradient.
* 

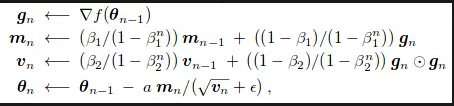
🡺 RMSprop mang lại một ưu điểm rõ ràng nhất, đó là khắc phục vấn đề tốc độ học giảm dần của Adagrad. Vấn đề này có thể khiến quá trình huấn luyện trở nên chậm dần, gây ra sự đóng băng không mong muốn. Tuy nhiên, RMSprop cũng có nhược điểm của riêng nó. Thuật toán này có thể hội tụ tới các điểm cực tiểu cục bộ thay vì điểm cực tiểu toàn cục như Momentum. Do đó, một giải pháp thường được sử dụng là kết hợp cả hai thuật toán, Momentum và RMSprop, để tạo ra thuật toán tối ưu hóa Adam. Phần sau sẽ trình bày về Adam và cách nó hoạt động.

1. Adam:

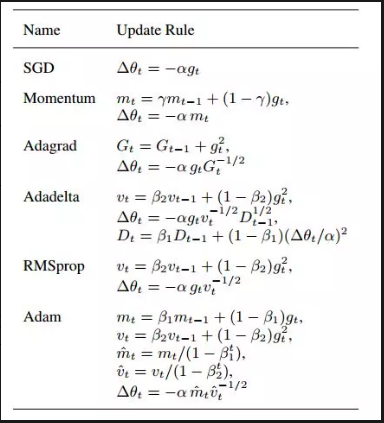
* Adam là sự kết hợp của Momentum và RMSprop . Nếu giải thích theo hiện tượng vật lí thì Momentum giống như 1 quả cầu lao xuống dốc, còn Adam như 1 quả cầu rất nặng có ma sát, vì vậy nó dễ dàng vượt qua local minimum tới global minimum và khi tới global minimum nó không mất nhiều thời gian dao động qua lại quanh đích vì nó có ma sát nên dễ dừng lại hơn.
* Giống với Adadelta và RMSprop, nó duy trì trung bình bình phương độ dốc (slope) quá khứ vt và cũng đồng thời duy trì trung bình độ dốc quá khứ mt, giống momentum.
* Trong khi momentum giống như một quả cầu lao xuống dốc, thì Adam lại giống như một quả cầu rất nặng và có ma sát (friction), nhờ vậy nó dễ dàng vượt qua local minimum và đạt tới điểm tối ưu nhất (flat minimum)
* Nó đạt được hiệu ứng Heavy Ball with Friction (HBF) nhờ vào hệ số (mt/ sqrt(vt))



* Công thức update của nó là:



* Tổng kết các công thức tối ưu:



1. Tìm hiểu về Continual Learning và Test Production khi xây dựng một giải pháp học máy để giải quyết một bài toán nào đó.
2. Continual Learning.
3. Continual Learning là gì?

* Học liên tục(Continual Learning) còn được gọi là học máy liên tục (CML), là quá trình mà một mô hình học từ các luồng dữ liệu mới mà không cần phải được huấn luyện lại.
* Khác với các phương pháp truyền thống, trong đó các mô hình được huấn luyện trên một bộ dữ liệu tĩnh, triển khai và định kỳ được huấn luyện lại, các mô hình học liên tục cập nhật lặp đi lặp lại các tham số của chúng để phản ánh các phân phối mới trong dữ liệu.
* Trong quá trình sau, mô hình cải thiện bản thân bằng cách học từ phiên bản mới nhất và cập nhật kiến thức của mình khi dữ liệu mới trở nên có sẵn. Vòng đời của mô hình học liên tục cho phép các mô hình duy trì tính phù hợp theo thời gian do chất lượng động bản chất của chúng.
* Một số yếu tố chính cần thiết để tạo ra một môi trường học tập liên tục. Chúng bao gồm những điều sau đây:

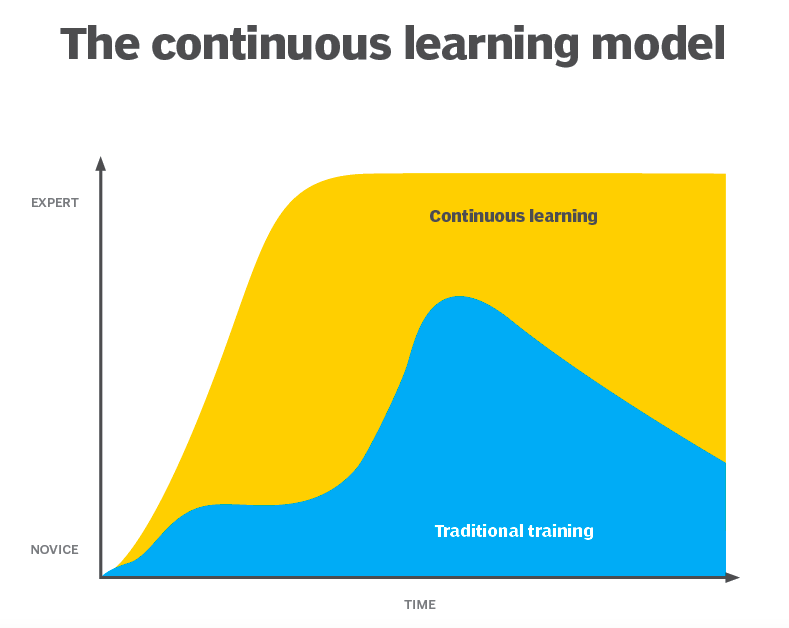
+ Cơ hội học tập dễ tiếp cận bất cứ khi nào có nhu cầu.

+ Cơ hội liên tục cho người học áp dụng kiến ​​thức và kiểm tra các kỹ năng mới của họ.

+ Một nền văn hóa thúc đẩy việc học tập bằng các phương pháp thực hành bền vững có thể lặp lại thành công.

+ Cơ hội hợp tác để người học chia sẻ kiến ​​thức và quan điểm.

+ Cơ chế phản hồi thường xuyên từ cả giảng viên và sinh viên.

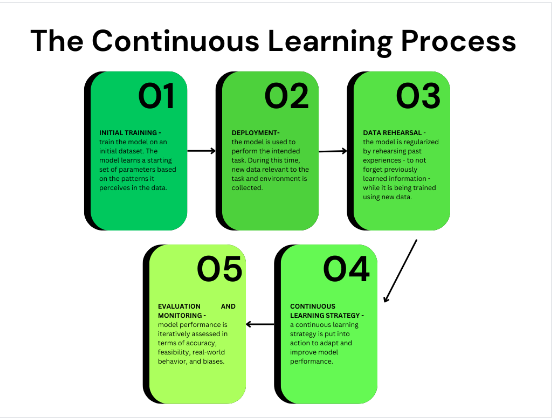


1. Các loại học máy liên tục:

* Có nhiều phương pháp học máy liên tục để xây dựng mô hình, và một số chiến lược phổ biến bao gồm học tập tăng dần, học tập chuyển giao và học tập suốt đời.
* Ngoài ra, có các phương pháp khác như phát lại trải nghiệm và kỹ thuật chính quy hóa. Tuy nhiên, việc lựa chọn phương pháp tiếp cận không phải là một quyết định đơn giản, mà phụ thuộc vào nhiều yếu tố như loại dữ liệu, kiến ​​trúc của mô hình, hiệu suất mong muốn, độ phức tạp của nhiệm vụ và tài nguyên tính toán có sẵn.
* Thường xuyên, sự kết hợp của các phương pháp tiếp cận được áp dụng để tối ưu hóa khả năng học tập của mô hình, đồng thời đáp ứng được các yếu tố cụ thể của môi trường và ứng dụng cụ thể.

1. Quá trình học tập liên tục:

* Học liên tục là một sự phát triển của mô hình học máy truyền thống. Do đó, nó liên quan đến nhiều nguyên tắc lập mô hình giống nhau: tiền xử lý, lựa chọn mô hình, điều chỉnh siêu tham số, đào tạo, triển khai và giám sát.
* Hai bước bổ sung cần thiết trong quá trình học tập liên tục: diễn tập dữ liệu và thực hiện chiến lược học tập liên tục. Các bước này nhằm đảm bảo rằng mô hình đang học từ các luồng dữ liệu mới một cách hiệu quả dựa trên ứng dụng và bối cảnh của nhiệm vụ dữ liệu.



Hình 1: Sơ đồ quy trình học tập liên tục đến học máy

1. Ưu điểm của việc học liên tục:

* Học liên tục có thể hữu ích cho tất cả các loại dự án dữ liệu: mô tả, chẩn đoán, dự đoán và quy định. Nó đặc biệt quan trọng trong các trường hợp liên quan đến dữ liệu thay đổi nhanh. Lợi ích so với phương pháp học máy truyền thống bao gồm:

+ Sự Khái Quát: Học liên tục giúp mô hình trở nên mạnh mẽ và chính xác hơn khi đối mặt với dữ liệu mới, cung cấp tính linh hoạt và khả năng tự điều chỉnh.

+ Lưu Giữ Thông Tin: Bằng cách áp dụng chiến lược học liên tục, mô hình sẽ duy trì thông tin từ những kiến thức trước đó, cho phép nó tích luỹ và cập nhật thông tin theo thời gian.

+ Khả Năng Thích Ứng: Mô hình sử dụng học liên tục sẽ linh hoạt và thích ứng tốt hơn với kiến thức mới, như sự thay đổi trong khái niệm hay xu hướng mới, giúp cải thiện khả năng dự đoán và linh hoạt trong dự báo về tương lai.

1. Nhược điểm của việc học tập liên tục:

* Một phương pháp học liên tục không thiếu những nhược điểm. Những điều này cần được xem xét trước khi chọn lựa một phương pháp như vậy dựa trên bản chất và kết quả dự kiến của nhiệm vụ hiện tại.
* Từ góc độ vận hành, một thách thức là:

+ Chi Phí: Các phương pháp học liên tục, mặc dù hiệu quả, thường phức tạp tính toán hơn so với các phương pháp truyền thống vì mô hình cần phải liên tục thích nghi với dữ liệu mới. Điều này thường dẫn đến chi phí kinh tế cao hơn vì đòi hỏi nhiều tài nguyên dữ liệu, nhân lực và máy tính.

* Từ góc độ mô hình hóa, một số hạn chế là:

+ Quản lý Mô hình: Mỗi khi tham số của một mô hình được cập nhật dựa trên dữ liệu mới, một mô hình mới được hình thành. Do đó, một phương pháp học liên tục có thể tạo ra một số lượng lớn mô hình, làm phức tạp việc xác định mô hình có hiệu suất tốt nhất.

+ Biến đổi Dữ liệu: Đối với một phương pháp học liên tục để đáng giá, chúng ta cần xử lý một lượng lớn dữ liệu mới. Tuy nhiên, một mô hình như vậy có nguy cơ mất đi khả năng dự đoán nếu phân phối đặc tính thay đổi đột ngột.

1. Học tập liên tục cho trí tuệ nhân tạo và học máy:

* Khái niệm học liên tục áp dụng cho hệ thống Trí tuệ nhân tạo (AI) và Học máy (ML). Học liên tục đóng vai trò quan trọng trong các hệ thống này. Hệ thống ML sử dụng các thuật toán để học cách tự phân tích dữ liệu. Các thuật toán này giúp chúng phân biệt thông tin quan trọng và đưa ra dự đoán từ thông tin đó.
* Trong quá trình học tĩnh, khi thuật toán ML được huấn luyện trên một tập dữ liệu cụ thể, nó giả định rằng tất cả các tập dữ liệu trong tương lai mà nó phân tích đều giống nhau. Tuy nhiên, thực tế và kiến thức không tĩnh lặng như vậy. Tương tự như cách con người phải liên tục học hỏi và nâng cao kỹ năng, hệ thống ML cũng cần trải qua quá trình đào tạo liên tục như một phần của quá trình hoạt động của nó.
* Mô hình ML được triển khai một lần, sau đó được theo dõi và đào tạo lại liên tục để thích ứng với dữ liệu thay đổi liên tục. Có nhiều kỹ thuật và công cụ khác nhau mà các nhà phát triển trong lĩnh vực này sử dụng để tự động hóa quá trình đào tạo lại này.
* Quá trình học liên tục cho ML đòi hỏi sự giám sát định kỳ từ các nhà phát triển con người. Quá trình đào tạo lại này cũng có những hạn chế vì yêu cầu cơ sở hạ tầng công nghệ đắt tiền và là quá trình tốn thời gian. Tuy nhiên, nó rất quan trọng để đảm bảo hiệu quả của hệ thống AI và ML.

1. Ứng dụng của việc học liên tục:

Với chi phí và độ phức tạp bổ sung xuất phát từ học liên tục, phương pháp này phù hợp nhất cho các ứng dụng liên quan đến một luồng dữ liệu liên tục. Điều này đòi hỏi môi trường của nhiệm vụ dữ liệu phải liên tục tiến triển và thay đổi.

Các ứng dụng hiện tại của học tập liên tục bao gồm:

* Thị giác máy tính. Tính động của dữ liệu dựa trên hình ảnh làm cho các phương pháp học liên tục trở nên phổ biến trong việc huấn luyện các thuật toán nhằm nhận diện và phân loại thông tin hình ảnh. Chúng thường được áp dụng trong việc nhận dạng khuôn mặt và công nghệ hình ảnh.
* An ninh mạng. Các phương pháp học liên tục được triển khai để đảm bảo giám sát liên tục trong cơ sở hạ tầng an ninh thông tin. Chúng là yếu tố quan trọng trong việc phát hiện lừa đảo, xâm nhập mạng và thư rác, trong số các hoạt động liên quan đến bảo mật.
* Chăm sóc sức khỏe. Do bản chất tiến triển của các bệnh tật, các phương pháp học liên tục được sử dụng trong nhiều lĩnh vực chăm sóc sức khỏe để nâng cao quy trình chẩn đoán bệnh tật. Các chuyên ngành như ung thư học và chẩn đoán hình ảnh đã sớm khám phá và sử dụng trí tuệ nhân tạo học liên tục với mục tiêu này.
* Robot. Học liên tục đã được sử dụng để cải thiện tính linh hoạt và hiệu suất của robot trong các môi trường khác nhau, giúp chúng tối ưu hóa hành động dựa trên kinh nghiệm mới và trước đó trong điều kiện thay đổi.

1. Test Production:
2. Test Production là gì ?

* Testi Production (TP) là một chiến lược trong đó việc kiểm thử phần mềm diễn ra trong môi trường sản xuất thực tế thay vì trong môi trường kiểm thử mô phỏng hoặc riêng biệt. Nó bao gồm việc triển khai các tính năng hoặc thay đổi trực tiếp vào môi trường sản xuất và sau đó kiểm thử chúng dưới điều kiện thực tế.
* TP là một cách để thu thập thông tin phản hồi và phản hồi về cách thức thay đổi ảnh hưởng đến người dùng thực tế, hiệu suất hệ thống và chức năng tổng thể trong môi trường thực tế. Nó cho phép quan sát cách hệ thống hoạt động trong môi trường thực tế với tương tác người dùng thực sự, dữ liệu và mô hình sử dụng thực tế.
* Tuy nhiên, việc kiểm thử trong môi trường sản xuất đòi hỏi kế hoạch cẩn thận, giám sát và các chiến lược giảm thiểu rủi ro để ngăn chặn các vấn đề tiềm ẩn hoặc gây gián đoạn đối với người dùng cuối. Các kỹ thuật như cờ tính năng, triển khai Canary, triển khai từ từ và các công cụ giám sát thường được sử dụng để giảm thiểu rủi ro và cho phép kiểm thử được kiểm soát trong môi trường thực tế mà không ảnh hưởng đến toàn bộ người dùng.
* Mặc dù có những lợi ích trong việc cung cấp thông tin thực tế, TP nên được tiếp cận một cách cẩn trọng để đảm bảo trải nghiệm người dùng và sự ổn định của hệ thống được duy trì trong suốt quá trình kiểm thử.

1. Lợi ích của Test Production:

* Test Production (TP) đặt hy vọng vào việc chắc chắn rằng tính năng sẽ hoạt động tốt trong môi trường mà nó được áp dụng. Quan trọng nhất, không phải ai quan tâm nếu tính năng hoạt động trong quá trình thử nghiệm; họ quan tâm khi nó được sử dụng trong môi trường sản xuất. Điều duy nhất để kiểm tra xem chúng có hoạt động trong môi trường sản xuất hay không chính là thử nghiệm trực tiếp tại đó.
* Chuyển sang TP cũng giúp QA tránh việc tốn công sức thiết lập một môi trường thử nghiệm riêng biệt. Luôn cần nỗ lực để tạo ra một môi trường thử nghiệm dựng phản ánh môi trường thực tế. Thử nghiệm tại nơi sẽ sử dụng giúp loại bỏ những công việc không cần thiết.
* Với sự liên kết chặt chẽ trong các hệ thống sản xuất hiện đại và nguy cơ rủi ro, việc thử nghiệm sản xuất đã trở thành một phần không thể thiếu của quá trình kiểm thử phần mềm và devops. Các công ty như Google, Netflix và Amazon thường xuyên thử nghiệm các tính năng mới (thậm chí chỉ với một phần nhỏ người dùng của họ) bằng cách sử dụng thử nghiệm trong môi trường sản xuất.

1. Tại sao cần Test Production?

* Có những ranh giới cần xem xét khi áp dụng Test Production (TP). Để thực hiện TP, mã sản xuất phải có khả năng phục hồi sau mọi rủi ro từ quá trình thử nghiệm. Mã có tính dễ vỡ hoặc ghi chép kém khiến việc áp dụng TiP trong môi trường sản xuất trở nên không an toàn.
* Nếu mã cần được kiểm tra bị hạn chế về phạm vi, TP có thể kéo dài thời gian kiểm tra. Ví dụ, nếu chỉ một phần của mã sản xuất bị ảnh hưởng bởi mã mới, việc đưa mã sản xuất về trạng thái thực thi chính xác để kiểm tra liên quan có thể tăng thêm số bước trong quy trình kiểm tra.
* Cân nhắc kỹ lưỡng về khả năng phục hồi của mã, và sự ảnh hưởng đến quá trình thực thi trong môi trường sản xuất là rất quan trọng trước khi quyết định áp dụng TP.

1. Một số mẹo khi Test Production:

Chia kiểm thử trên production thành các tầng:

* Trong khi chúng ta nói về ý tưởng 'kiểm thử trên môi trường production', nó bao gồm việc kiểm thử các ứng dụng chạy riêng trên nền tảng production, chạy trực tiếp các thử nghiệm với mã được triển khai 100% và kiểm thử toàn bộ danh sách test server trong trung tâm dữ liệu production.
* Do đó, việc kiểm tra trên môi trường production nên được chia thành các tầng để kiểm tra các khía cạnh khác nhau của môi trường production sản xuất theo những cách khác nhau.

Lên kế hoạch kiểm thử ở thời điểm mà người dùng ít sử dụng:

* Kiểm thử hiệu năng có thể có gây ra tác động lớn đến toàn bộ cơ sở người dùng. Nó có thể làm cho môi trường máy chủ không hoạt động, đó là điều không ai muốn. Chúng ta nên nghiên cứu các phân tích và xác định khi nào là thời điểm tốt nhất để lên lịch kiểm thử trên môi trường production.

Thu thập dữ liệu lưu lượng truy cập gốc cho việc kiểm thử:

* Chúng ta nên thu thập và sử dụng dữ liệu lưu lượng truy cập thực tế trong môi trường production (như các quy trình làm việc của người dùng, các tài nguyên và hành vi người dùng) để thúc đẩy việc tạo tải cho các testcase. Khi bạn thực hiện các kiểm thử trong môi trường production, bạn sẽ có 1 sự tự tin là hành vi mô phỏng đó là thật.

Tập trung giám sát:

* Trong khi chạy 1 thử nghiệm trên production, hãy luôn luôn để mắt tới các số liệu về hiệu năng của người dùng để biết được rằng liệu việc thử nghiệm có gây ra những ảnh hưởng không thể chấp nhận đến trải nghiệm người dùng hay không. Chuẩn bị tinh thần để ngừng việc kiểm thử nếu điều đó xảy ra.

Tạo 1 trải nghiệm “Opt-in”:

* Một cách tuyệt vời để kiểm tra ứng dụng sẽ hoạt động như thế nào với người dùng thực tế là có một số “opt-in” ở trong các bản phát hành tính năng mới. Điều này sẽ cho phép người dùng theo dõi và thu thập dữ liệu từ người dùng trong thời gian thực và thực hiện các điều chỉnh phù hợp với chiến lược kiểm thử nghiệm mà không lo ảnh hưởng tới trải nghiệm của họ.

1. Kết luận :

* Kiểm thử ứng dụng luôn được ưu tiên hàng đầu đối với các tổ chức phát triển phần mềm. Hầu hết họ tập trung vào việc kiểm thử càng nhiều càng tốt trước khi triển khai lên môi trường production để đảm bảo quá trình chuyển đổi sau triển khai diễn ra một cách mượt mà nhất có thể.
* Tuy nhiên, kiểm thử trên môi trường production mang lại những lợi ích độc đáo mà các tổ chức DevOps không nên bỏ qua. Từ việc chuẩn bị đội ngũ để xử lý các tình huống nguy hiểm trong môi trường production đến việc cải thiện trải nghiệm người dùng cho khách hàng, kiểm thử trên môi trường sản xuất ngày càng trở thành một phần không thể thiếu trong quá trình kiểm thử ứng dụng.
* Việc kiểm thử trên môi trường production cần được thực hiện như một phần của quy trình kiểm thử, được mở rộng và có khả năng khôi phục cao. Đồng thời, chúng ta cần tiếp tục thực hiện kiểm thử sớm và thường xuyên, và cân nhắc việc kiểm thử trên môi trường production (shift-right testing) là một phần của chiến lược kiểm thử.
* Mặc dù kiểm thử trên môi trường production có thể gặp nhiều rủi ro, nếu không thể tránh khỏi việc kiểm thử trên môi trường sản xuất, cần phải tiến hành một cách cẩn thận.

# **Tài Liệu Tham Khảo**

Trần Trung Trực(17/10/2020), “Optimizer- Hiểu sâu về các thuật toán tối ưu ( GD,SGD,Adam,..)”. Truy cập tại: <https://viblo.asia/p/optimizer-hieu-sau-ve-cac-thuat-toan-toi-uu-gdsgdadam-Qbq5QQ9E5D8#_5-rmsprop-6>

Phan Ngoc(27/05/2018), “Thuật toán tối ưu adam”, Truy cập tại: <https://viblo.asia/p/thuat-toan-toi-uu-adam-aWj53k8Q56m>

Tong Zhang(), “Adaptive Subgradient Methods for Online Learning and Stochastic Optimization”, Truy cập tại: <https://www.jmlr.org/papers/volume12/duchi11a/duchi11a.pdf>

Cafedev(21/12/2023), “Tự học ML | Giới thiệu Trình tối ưu hóa Gradient dựa trên Momentum”. Truy cập tại: <https://cafedev.vn/tu-hoc-ml-gioi-thieu-trinh-toi-uu-hoa-gradient-dua-tren-momentum/>

Quy Nguyen(03/04/2021), “Gradient Descent”. Truy cập tại: <https://ndquy.github.io/posts/gradient-descent-2/#stochastic-gradient-descent>

Datacamp(09/2023), “What is Continuous Learning? Revolutionizing Machine Learning & Adaptability”. Truy cập tại: <https://www.datacamp.com/blog/what-is-continuous-learning>

Cameron Hashemi-Pour(11/2023), “continuous learning”. Truy cập tại: <https://www.techtarget.com/whatis/definition/continuous-learning>

Ivan Andreev(17/06/2023), “continuous learning”. Truy cập tại: <https://www.valamis.com/hub/continuous-learning>

Thi Nguyen Thi(20/03/2020), ”Tại sao bạn nên kiểm thử trên môi trường Production?”. Truy cập tại: <https://viblo.asia/p/tai-sao-ban-nen-kiem-thu-tren-moi-truong-production-WAyK8oOm5xX>

Scott Fitzpatrick(16/10/2018), “The What and Why of Testing in Production”. Truy cập tại: <https://saucelabs.com/resources/blog/the-what-and-why-of-testing-in-production>

Larry Loeb(07/07/2021), “Why You Should Be Testing in Production”. Truy cập tại: <https://saucelabs.com/resources/blog/why-you-should-be-testing-in-production>