Dự án Cuối kỳ
Môn: Nhập môn Học Máy

Họ và Tên: Nguyễn Quốc Anh
MSSV: 52100871

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.

Bài làm

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

Mục tiêu chung trong huấn luyện các mô hình học máy là tìm ra được tham số phù hợp nhất cho thuật toán học máy. Do đó ta cần các phương pháp để tìm ra các tham số đó hiệu quả đó chính là các Optimizer. Các optimizer đều có cách hoạt động quay quanh việc đạo hàm các hàm loss theo trọng số từ đó có thể tìm được cực tiểu của hàm loss.

Vậy tóm lại Optimizer là một thuật toán được sử dụng để cập nhật các tham số của một mô hình học máy trong quá trình huấn luyện. Optimizer được sử dụng để tìm ra một tập hợp tham số tối ưu cho mô hình, là tập hợp tham số có thể dự đoán tốt nhất dữ liệu huấn luyện.

Và sau đây là các optimizers phổ biến hiện nay:

- Gradient Descent
- Stochastic Gradient Descent
- Momentum
- Adagrad
- RMSprop
- Adam

Ta sẽ cùng tìm hiểu kĩ hơn về các optimizer đó như sau:

1. **Gradient Descent (GD):**

- GD là thuật toán optimizer đơn giản và cơ bản nhất dựa trên đạo hàm của hàm loss.
- GD cập nhật các tham số theo hướng giảm dần của gradient của hàm loss.
- Ưu điểm:
    - Dễ hiểu và triển khai
- Nhược điểm:
    - Có thể bị kẹt ở cực tiểu cục bộ
    - Thời gian học chậm do thường phải chọn learning rate thấp
- Công thức:

$$w \leftarrow w - \eta \frac{\partial L}{\partial w}$$

Trong đó:

- w là trọng số
- η là learning rate
- L là hàm loss
- $\frac{\partial L}{\partial w}$ đại diện cho độ dốc tính được theo trọng số

2. **Stochastic Gradient Descent (SGD):**

- SGD hoạt động giống như với GD nhưng thay vì cập nhật trọng số sau mỗi epoch thì SGD cập nhật trọng số sau mỗi batch.
- SGD có mối quan hệ chặt chẽ với "batching".
- "Batching" là một quá trình tách dữ liệu thành nhiều "batch" và mình sẽ cập nhật trọng số sau mỗi "batch".
- Nhờ vào cơ chế này mà SGD sẽ đẩy nhanh quá trình đi về điểm cực tiểu hơn
- Nhưng cũng vì vậy mà sẽ bị không ổn định.

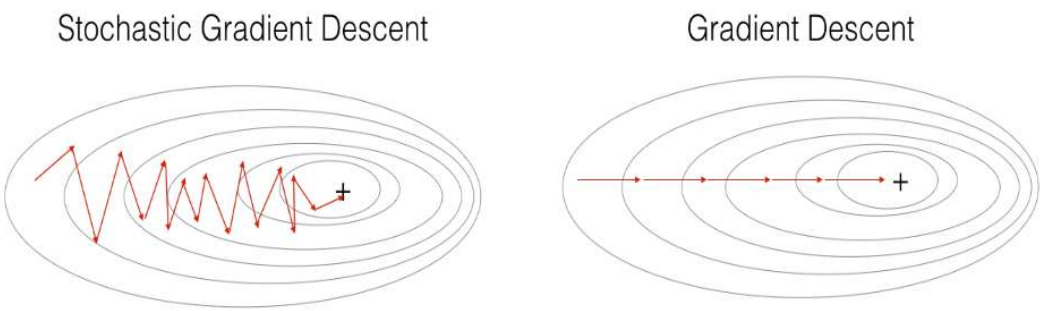


Figure 1 : **SGD vs GD**
"+" denotes a minimum of the cost. SGD leads to many oscillations to reach convergence. But each step is a lot faster to compute for SGD than for GD, as it uses only one training example (vs. the whole batch for GD).

- Thuật toán SGD:

Algorithm 8.1 Stochastic gradient descent (SGD) update

Require: Learning rate schedule $\epsilon_1, \epsilon_2, \ldots$
Require: Initial parameter $\boldsymbol{\theta}$
  $k \leftarrow 1$
  while stopping criterion not met **do**
    Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient estimate: $\hat{\boldsymbol{g}} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon_k \hat{\boldsymbol{g}}$
    $k \leftarrow k + 1$
  end while

3. Momentum:

- Vì nhược điểm lớn nhất của SGD và GD là ở việc rất dễ bị mắc lại ở các cực tiểu cục bộ
- Vì thế mà Momentum ra đời như một giải pháp của nhược điểm trên
- Momentum thực chất vẫn hoạt động trên SGD và GD nhưng được kế thừa và phát triển thêm. Có thể hình dung Momentum qua biểu đồ sau:

- Momentum (Quán tính): Bằng cách tưởng tượng việc trọng số được cập nhật là trái banh đang lăn xuống thì với quán tính có thể dễ dàng vượt qua các cực tiểu cục bộ để đến được cực tiểu toàn cục.
- Vậy cách tốt nhất để tìm được quán tính phù hợp cho trọng số là kiểm tra xem trọng số thay đổi như thế nào so với lần trước đó

- Công thức:

$$w \leftarrow w(t) - \eta \frac{\partial L}{\partial w}(t) - \alpha \eta \frac{\partial L}{\partial w}(t-1)$$

Trong đó:

- w là trọng số
- η là learning rate
- L là hàm loss
- $\frac{\partial L}{\partial w}(t)$ đại diện cho độ dốc tính được theo trọng số tại epoch (t)
- α là một hệ số Momentum (Là một hyperparameter ta có thể cho là một con số bất kỳ thường là 0.9)

- Thuật toán

Algorithm 8.2 Stochastic gradient descent (SGD) with momentum

Require: Learning rate ϵ, momentum parameter α
Require: Initial parameter $\boldsymbol{\theta}$, initial velocity $\boldsymbol{v}$
  while stopping criterion not met **do**
    Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient estimate: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$.
    Compute velocity update: $\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \boldsymbol{g}$.
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$.
  end while

Trước khi tìm hiểu các optimizer "Adagrad", "RMSprop" và "Adam" ta hãy thảo luận trước về việc thiết lập các "Learning Rate" trong các "optimizer"

Đặc điểm của "Learning Rate":

- "Small Enough": đủ nhỏ để có thể giảm từ từ thay vì dao động quá lớn
- "Big Enough": để mình có thể đến được giá trị cực tiểu trong thời gian sớm nhất

Vậy để thoả mãn các đặc tính trên của "learning rate" ta có thể sử dụng một "Learning Rate Schedule"

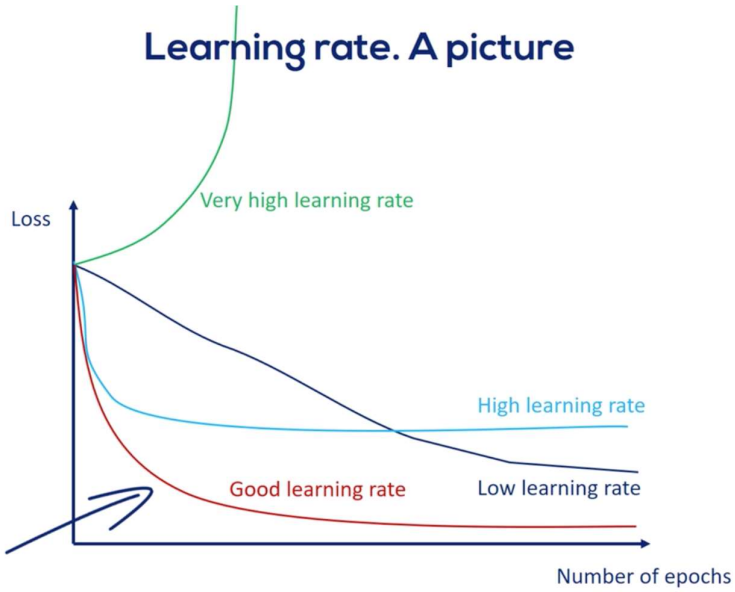
1. Ta có thể bắt đầu bằng một khởi tạo learning rate cao
2. Sau đó tại một điểm nào đó ta sẽ giảm learning rate lại để tránh dao động
3. Gần cuối ta sẽ chọn một learning rate cực kì thấp để có thể đạt được cực tiểu tốt nhất

Ta có thể mô hình hoá "Learning Rate" qua công thức sau:

$$\eta = \eta_0 e^{\frac{-n}{c}}$$

Trong đó:

- η là "Learning Rate"
- n là epoch hiện tại
- c là một hằng số phân rã mục đích để giảm η sau mỗi lần epoch



Vậy thì phương pháp optimizer tốt là một phương pháp có một "Learning Rate" tốt.

4. AdaGrad (Adaptive Gradient Algorithm):

- AdaGrad sẽ cập nhật learning rate thích ứng theo trọng số được thay đổi sau mỗi lần cập nhật.
- Các trọng số có độ dốc cao hoặc cập nhật thường xuyên sẽ có learning rate thấp lại để tránh việc vượt qua cực tiểu
- Ngược lại với các trọng số có độ dốc thấp hoặc cập nhật không thường xuyên sẽ có learning rate cao để có thể giảm tới mức cực tiểu nhanh hơn
- Công thức:

$$w(t+1) = w(t) + \Delta w(t)$$

$$\Delta w_i(t) = -\frac{\eta}{\sqrt{G_i(t)} + \varepsilon} \frac{\partial L}{\partial w_i}(t)$$

$$G_i(t) = G_i(t-1) + \left(\frac{\partial L}{\partial w_i}(t)\right)^2$$

Trong đó:

- G là hàm đóng vai trò giảm learning rate xuống ($G_i(0) = 0$) và hàm G luôn tăng do đó "Learning Rate" luôn giảm
- ε là một hằng số rất nhỏ để tránh trường hàm chia cho 0

Thuật toán:

Algorithm 8.4 The AdaGrad algorithm

Require: Global learning rate ϵ
Require: Initial parameter $\boldsymbol{\theta}$
Require: Small constant δ, perhaps 10^{-7}, for numerical stability
Initialize gradient accumulation variable $\boldsymbol{r} = \mathbf{0}$
while stopping criterion not met **do**
Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$.
Accumulate squared gradient: $\boldsymbol{r} \leftarrow \boldsymbol{r} + \boldsymbol{g} \odot \boldsymbol{g}$.
Compute update: $\Delta \boldsymbol{\theta} \leftarrow -\frac{\epsilon}{\delta + \sqrt{\boldsymbol{r}}} \odot \boldsymbol{g}$. (Division and square root applied element-wise)
Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta \boldsymbol{\theta}$.
end while

5. RMSProp (Root mean square propagation)

- RMSProp là một cải tiến nhỏ của AdaGrad với sự giới hạn lịch sử của các độ dốc trước đó để tránh trường hợp "learning rate" bị giảm quá nhanh
- Công thức:

$$w(t+1) = w(t) + \Delta w(t)$$

$$\Delta w_i(t) = -\frac{\eta}{\sqrt{G_i(t)} + \varepsilon} \frac{\partial L}{\partial w_i}(t)$$

$$G_i(t) = \beta G_i(t-1) + (1-\beta) \left(\frac{\partial L}{\partial w_i}(t) \right)^2$$

Trong đó:

- β là hệ số giảm giữ lại thông tin trước đó.

Thuật toán:

Algorithm 8.5 The RMSProp algorithm

Require: Global learning rate ϵ, decay rate ρ
Require: Initial parameter $\boldsymbol{\theta}$
Require: Small constant δ, usually 10^{-6}, used to stabilize division by small numbers
Initialize accumulation variables $\boldsymbol{r} = 0$
while stopping criterion not met **do**
Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$.
Accumulate squared gradient: $\boldsymbol{r} \leftarrow \rho \boldsymbol{r} + (1 - \rho) \boldsymbol{g} \odot \boldsymbol{g}$.
Compute parameter update: $\Delta \boldsymbol{\theta} = -\frac{\epsilon}{\sqrt{\delta + \boldsymbol{r}}} \odot \boldsymbol{g}$. ($\frac{1}{\sqrt{\delta + \boldsymbol{r}}}$ applied element-wise)
Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta \boldsymbol{\theta}$.
end while

6. Adam (Adaptive Moment Estimation):

- Adam là sự kết hợp hoàn hảo giữa RMSProp và Momentum. Nó có khả năng thay đổi learning rate thích ứng cho từng tham số cũng như có thêm "quán tính" để vượt qua các cực tiểu cục bộ
- Cùng quay lại với 2 công thức của RMSProp và Momentum:

RMSProp: $\Delta w_i(t) = -\frac{\eta}{\sqrt{G_i(t) + \varepsilon}} \frac{\partial L}{\partial w_i}(t)$

Momentum: $w \leftarrow w(t) - \eta \frac{\partial L}{\partial w}(t) - \alpha \eta \frac{\partial L}{\partial w}(t-1)$

Adam sẽ kế thừa hai công thức trên:

$$\Delta w_i(t) = -\frac{\eta}{\sqrt{G_i(t) + \varepsilon}} M_i(t)$$

$$M_i(t) = \alpha M_i(t-1) + (1 - \alpha) \frac{\partial L}{\partial w_i}(t)$$

$$M_i(0) = 0$$

Trong đó:

- α là một hệ số Momentum.

Algorithm 8.6 RMSProp algorithm with Nesterov momentum

Require: Global learning rate ϵ, decay rate ρ, momentum coefficient α

Require: Initial parameter $\boldsymbol{\theta}$, initial velocity $\boldsymbol{v}$

Initialize accumulation variable $\boldsymbol{r} = \boldsymbol{0}$

while stopping criterion not met **do**

Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.

Compute interim update: $\tilde{\boldsymbol{\theta}} \leftarrow \boldsymbol{\theta} + \alpha \boldsymbol{v}$.

Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\tilde{\boldsymbol{\theta}}} \sum_i L(f(\boldsymbol{x}^{(i)}; \tilde{\boldsymbol{\theta}}), \boldsymbol{y}^{(i)})$.

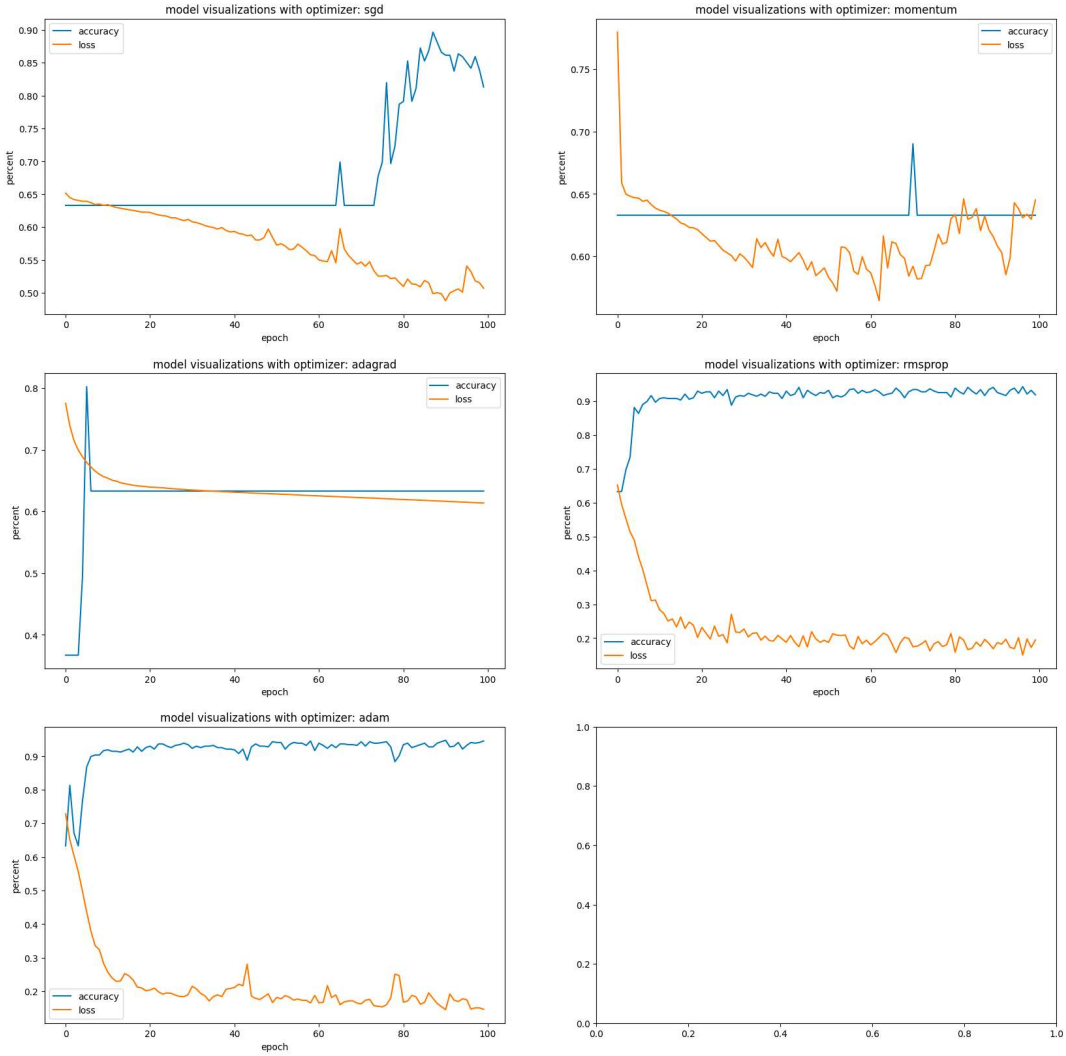
Accumulate gradient: $\boldsymbol{r} \leftarrow \rho \boldsymbol{r} + (1 - \rho) \boldsymbol{g} \odot \boldsymbol{g}$.

Compute velocity update: $\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \frac{\epsilon}{\sqrt{\boldsymbol{r}}} \odot \boldsymbol{g}$. ($\frac{1}{\sqrt{\boldsymbol{r}}}$ applied element-wise)

Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$.

end while

Bảng so sánh chung các optimizer qua dataset breast_cancer và sử dụng Feed Forward Neural Network:

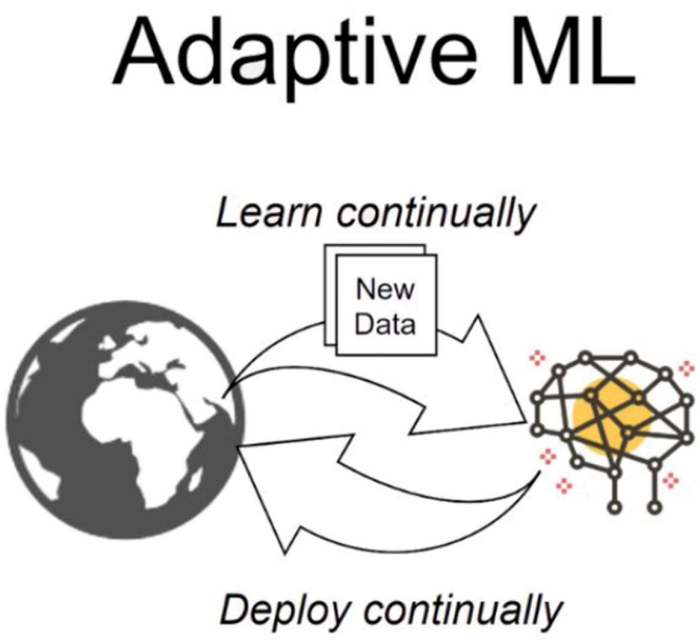


2) Tìm hiểu về Continual Learning và Test Production khi xây dựng một giải pháp học máy để giải quyết một bài toán nào đó.

1. Continual Learning

a. Khái niệm.

Continual Learning (CL) là một lĩnh vực nghiên cứu của học máy tập trung vào việc phát triển các mô hình học máy có thể học hỏi và thích ứng với dữ liệu mới trong khi vẫn giữ được hiệu suất trên dữ liệu cũ. Điều này là cần thiết trong nhiều ứng dụng thực tế, nơi dữ liệu liên tục thay đổi theo thời gian.



b. Tại sao cần phải có Continual Learning?

Dữ liệu luôn thay đổi: Thế giới luôn thay đổi, và dữ liệu mà chúng ta sử dụng để đào tạo các mô hình học máy cũng vậy. Nếu chúng ta không cập nhật các mô hình của mình với dữ liệu mới, chúng sẽ không thể hoạt động hiệu quả trong thế giới thực.

Các mô hình học máy có thể được cải thiện: Các kỹ thuật học máy luôn được cải thiện. Nếu chúng ta không cập nhật các mô hình của mình với các kỹ thuật mới, chúng sẽ không thể tận dụng được những cải tiến này.

Tăng trải nghiệm người dùng: Khi mô hình học máy của ta được học tiếp tục với dữ liệu mới từ người dùng thì khi đó dữ liệu của ta sẽ sát với người dùng hơn từ đó tăng trải nghiệm người dùng hơn.

2. Test Production

a. Khái niệm

Test Production là một quá trình trong đó một giải pháp học máy được triển khai trong môi trường sản xuất, nhưng vẫn được kiểm tra và giám sát chặt chẽ. Điều này cho phép các nhà phát triển phát hiện và giải quyết bất kỳ vấn đề nào với giải pháp trước khi nó ảnh hưởng đến người dùng.

b. Lợi ích

Test Production mang lại nhiều lợi ích cho giải pháp học máy, bao gồm:

- **Tăng hiệu suất:** Test Production giúp phát hiện và giải quyết các vấn đề hiệu suất trước khi chúng ảnh hưởng đến người dùng. Điều này giúp đảm bảo giải pháp hoạt động tốt trong môi trường sản xuất.
- **Tăng khả năng mở rộng**: Test Production giúp phát hiện và giải quyết các vấn đề mở rộng trước khi chúng ảnh hưởng đến người dùng. Điều này giúp đảm bảo giải pháp có thể đáp ứng nhu cầu của người dùng trong môi trường sản xuất.