**TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM**

**TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG**

**KHOA CÔNG NGHỆ THÔNG TIN**



**FINAL PROJECT IN MACHINE LEARNING**

**Các phương pháp Optimizer, Continual Learning và Test Production khi xây dựng một giải pháp học máy**

Người thực hiện: **NGUYỄN THÁI KHÔI – MSSV: 52100637**

**THÀNH PHỐ HỒ CHÍ MINH, NĂM 2023**

**MỤC LỤC**

Contents

[**I.** **CÁC PHƯƠNG PHÁP OPTIMIZERS TRONG HUẤN LUYỆN MÔ HÌNH HỌC MÁY.** 3](#_Toc154009204)

[**1.** **STOCHASTIC GRADIENT DESCENT (SGD)** 3](#_Toc154009205)

[**1.1.** **KHÁI NIỆM CƠ BẢN VỀ SGD** 3](#_Toc154009206)

[**1.2.** **SGD OPTIMIZERS TRONG KERAS** 3](#_Toc154009207)

[**2.** **ADAPTIVE GRADIENT DESCENT (AdaGrad)** 4](#_Toc154009208)

[**2.1.** **KHÁI NIỆM CƠ BẢN VỀ AdaGra** 4](#_Toc154009209)

[**2.2.** **AdaGrad TRONG KERAS** 5](#_Toc154009210)

[**3.** **Adadelta OPTIMIZER** 5](#_Toc154009211)

[**3.1.** **KHÁI NIỆM CƠ BẢN VỀ Adadelta** 5](#_Toc154009212)

[**3.2.** **Adadelta TRONG KERAS** 6](#_Toc154009213)

[**4.** **RMSprop OPTIMIZER** 6](#_Toc154009214)

[**4.1.** **KHÁI NIỆM CƠ BẢN VỀ RMSprop** 6](#_Toc154009215)

[**4.2.** **RMSprop TRONG KERAS** 6](#_Toc154009216)

[**5.** **ADAPTIVE MOMENT ESTIMATION OPTIMIZER (Adam)** 7](#_Toc154009217)

[**5.1.** **KHÁI NIỆM CƠ BẢN VỀ Adam** 7](#_Toc154009218)

[**5.2.** **Adam OPTIMIZER TRONG KERAS** 7](#_Toc154009219)

[**II.** **CONTINUAL LEARNING AND TEST IN PRODUCTION** 8](#_Toc154009220)

[**1.** **Continual Learning** 8](#_Toc154009221)

[**1.1.** **Continual Learning là gì?** 8](#_Toc154009222)

[**1.2.** **Tại sao lại cần Continual Learning?** 8](#_Toc154009223)

[**1.3.** **Stateless Retraining và Stateful Training** 8](#_Toc154009224)

[**1.4.** **Bốn giai doạn của Continual Learning** 9](#_Toc154009225)

[**2.** **Test In Production** 9](#_Toc154009226)

[**1.** **Shadow Deployment** 10](#_Toc154009227)

[**2.** **A/B Testing** 10](#_Toc154009228)

[**3.** **Canary Release** 11](#_Toc154009229)

[**4.** **Interleaving Experiments** 11](#_Toc154009230)

[**5.** **Bandits** 12](#_Toc154009231)

# **CÁC PHƯƠNG PHÁP OPTIMIZERS TRONG HUẤN LUYỆN MÔ HÌNH HỌC MÁY.**

## **STOCHASTIC GRADIENT DESCENT (SGD)**

### **KHÁI NIỆM CƠ BẢN VỀ SGD**

Stochastic Gradient Descent là một biến thể của Gradient Descent, nó giúp cải thiện hiệu suất của Gradient Descent khi giải quyết một bài toán với dataset lớn.

Trong SGD, thay vì sử dụng toàn bộ dataset để tính gradient cho mỗi vòng lặp như Gradient Descent, chỉ một phần nhỏ dataset được chọn một cách ngẫu nhiên để tính gradient và cập nhật giá trị tham số.

Công thức của SGD được thể hiện như sau:

Trong đó:

* : là tham số của mô hình.
* : là learning rate.
* : là hàm mất mát của mô hình.
* là các điểm dữ liệu được chọn một cách ngẫu nhiên trong dataset.

Ưu điểm của Stochastic Gradient Descent:

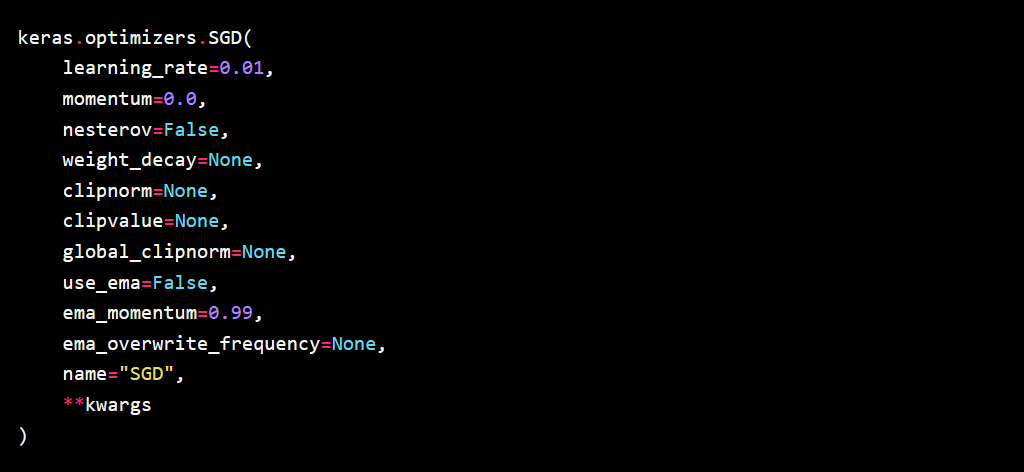
* Với sự thay đổi này, hiệu suất tính toán được cải thiện đáng kể (khi thực hiện với dataset lớn) so với Gradient Descent thông thường.

Nhược điểm của Stochastic Gradient Descent:

* Các điểm dữ liệu ngẫu nhiên không thể đại diện được cho toàn bộ dữ liệu trong mỗi vòng lặp.

### **SGD OPTIMIZERS TRONG KERAS**

Trong thư viện TensorFlow, phương pháp optimizer SGD có thể được import thông qua *tensorflow.keras.optimizers.SGD.*



Một số tham số cơ bản:

* **Learning\_rate:** Learning rate của thuật toán.

## **ADAPTIVE GRADIENT DESCENT (AdaGrad)**

### **KHÁI NIỆM CƠ BẢN VỀ AdaGra**

Trong khi Gradient Descent và các biến thể khác sử dụng một hệ số learning rate cố định trong suốt quá trình training, AdaGrad có learning rate được cập nhật qua mỗi vòng lặp trong quá trình training.

Phương pháp optimizer AdaGrad có công thức như sau:

Trong đó:

* là learning rate ban đầu.
* là một số dương rất nhỏ, được dùng để tránh tình trạng mẫu số bằng 0.
* được tính theo công thức sau đây:

**Ưu điểm của AdaGrad:**

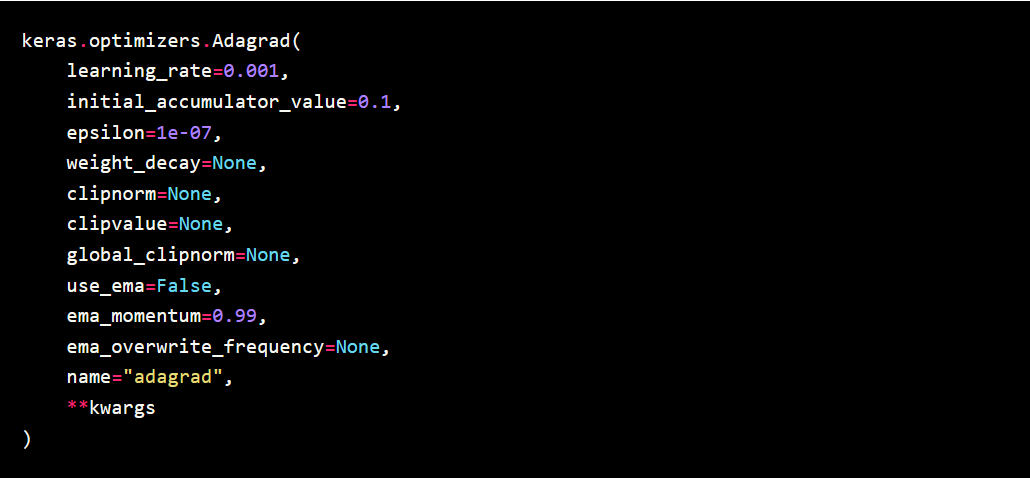
* Learning rate được cập nhật tự động.
* Cho ra kết quả tốt hơn SGD, vì learning được cập nhật phù hợp với tình trạng của bài toán.

**Nhược điểm của AdaGrad:**

* Bởi vì được cộng thêm một lượng bằng **,** vì thế learning rate sẽ luôn luôn giảm và có thể bằng một số cực kì nhỏ, khiến cho việc hội tụ xảy ra chậm hơn (hoặc có thể không hội tụ).

### **AdaGrad TRONG KERAS**

Trong thư viện Tensorflow, phương pháp Optimizer AdaGrad có thể được import thông qua *tensorflow.keras.optimers.AdaGrad*.



Một số tham số cơ bản:

* **Learning\_rate:** (*float)* Learning rate của thuật toán. Giá trị mặc định bằng 0.001.
* **epsilon:** Một giá trị dương rất nhỏ, được thêm vào phương trình để tránh việc mẫu số bằng 0.

## **Adadelta OPTIMIZER**

### **KHÁI NIỆM CƠ BẢN VỀ Adadelta**

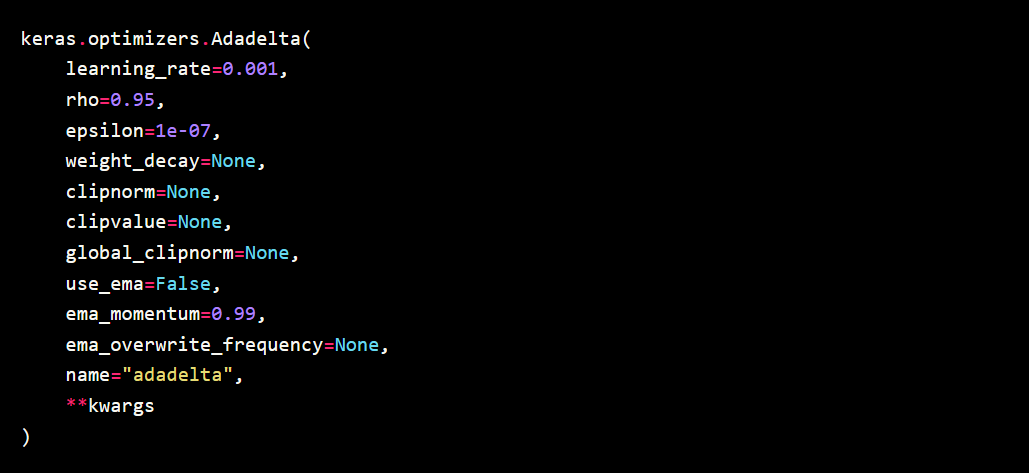
Adadelta là một phương pháp mở rộng của AdaGrad optimizer, Adadelta giúp làm giảm sự giảm nhanh chóng và liên tục của learning rate trong AdaGrad.

Thay vì sử dụng như AdaGrad, phương pháp Adadelta sử dụng là tổng của toàn bộ đạo hàm bình phương trước đó của **J**, được tính theo công thức sau:

Tham số trong phương pháp Adadelta được cập nhật theo công thức sau:

### **Adadelta TRONG KERAS**

Trong thư viện Tensorflow, Adadelta có thể được import thông qua *tensorflow.keras.optimizers.Adadelta*.



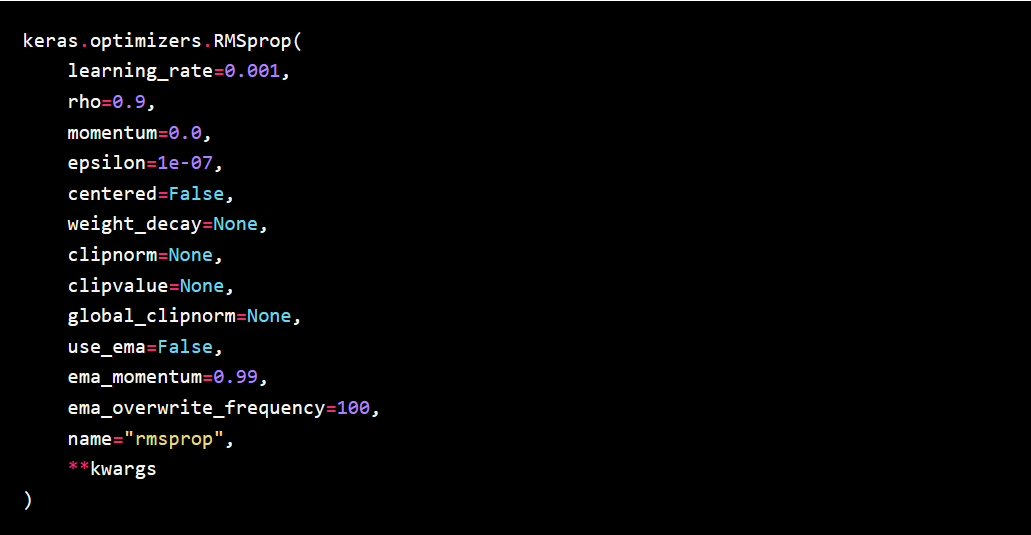
## **RMSprop OPTIMIZER**

### **KHÁI NIỆM CƠ BẢN VỀ RMSprop**

Phương pháp Optimizer RMSprop có công thức tương tự như Adadelta, tuy nhiên thay vì sử dụng tham số , RMSprop xác định là 2 hằng số 0.9 và 0.1. Công thức sau đây sẽ giúp hiểu rõ hơn RMSprop:

### **RMSprop TRONG KERAS**

Trong thư viện Tensorflow, RMSprop có thể được import thông qua *tensorflow.keras.RMSprop*.



## **ADAPTIVE MOMENT ESTIMATION OPTIMIZER (Adam)**

### **KHÁI NIỆM CƠ BẢN VỀ Adam**

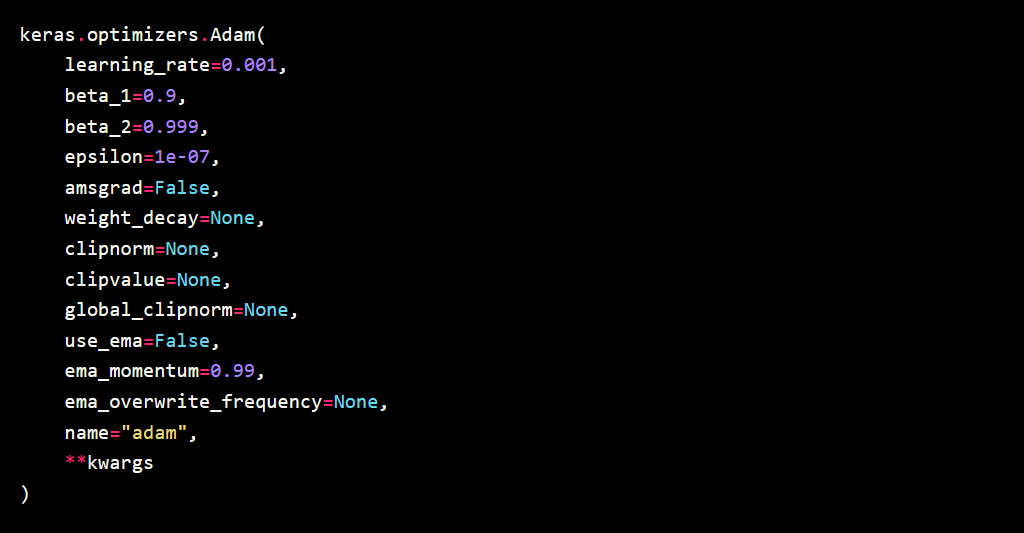
Adaptive Moment Estimation (Adam) là một phương pháp tính learning rate phù hợp cho từng tham số.

Ngoài việc lưu trữ giá trị trung bình giảm dần theo cấp số nhân của gradient bình phương trong quá khứ (giống như 2 phương pháp Adadelta và RMSprop), Adam cũng giữ mức trung bình giảm dần theo cấp số nhân của độ dốc trong quá khứ , tương tự như động lượng.

Tham số được cập nhật theo công thức sau:

### **Adam OPTIMIZER TRONG KERAS**

Trong thư viện Tensorflow, thuật toán tối ưu Adam có thể được import thông qua *tensorflow.keras.Adam.*



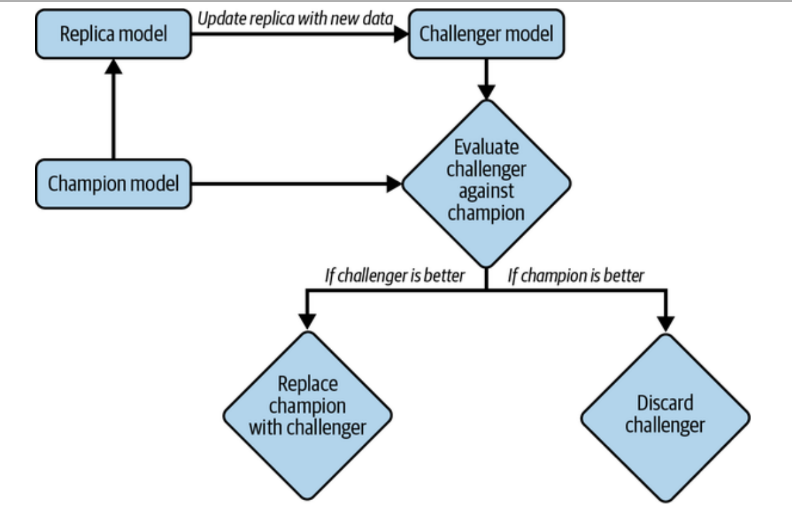
# **CONTINUAL LEARNING AND TEST IN PRODUCTION**

## **Continual Learning**

### **Continual Learning là gì?**

Continual Learning là một giải pháp giúp cập nhật mô hình Machine Learning với dữ liệu mới.

Continual Learning không có nghĩa là chúng ta phải re-train lại model mỗi khi có một điểm dữ liệu mới xuất hiện, điều này rất nguy hiểm, đặc biệt đối với neural network bởi vì nó khiến neural network bị “catastrophic forgetting”.



### **Tại sao lại cần Continual Learning?**

Continual Learning được áp dụng trong một số trường hợp sau đây:

* Trong trường hợp thay đổi một cách nhanh chóng và bất ngờ của dữ liệu.
* Trong trường hợp không thể lấy được training data cho một sự kiện cụ thể.
* Trong trường hợp model dự đoán một dữ kiện mới, tuy nhiên không có data của dữ kiện đó trong quá khứ (hoặc data đã quá cũ).

### **Stateless Retraining và Stateful Training**

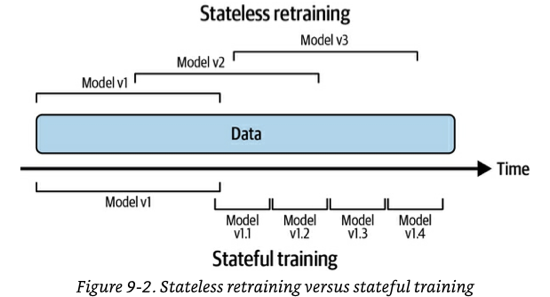
* + 1. **Stateless Retraining**

Retrain lại model từ đầu mỗi khi có dữ liệu mới, sử dụng trọng số ngẫu nhiên và một dữ liệu hoàn toàn mới (dữ liệu có thể trùng lắp với dữ liệu cũ được sử dụng cho việc training model trước đó).

* + 1. **Statefull Training**

Statefull Training sẽ khởi tạo một model với trọng số của model trước đó và tiếp tục sử dụng model mới để tranining dữ liệu mới.

Statefull Training cho phép model update dữ liệu kể cả chỉ có một lượng rất nhỏ dữ liệu mới.



### **Bốn giai doạn của Continual Learning**

**Giai đoạn 1: Stateless Retraining**

Model chỉ được retrain khi đáp ứng đủ 2 điều kiện sau đây:

* Hiệu suất của model giảm một cách đáng kể.
* Chúng ta có thời gian để retrain lại model.

**Giai đoạn 2: Lên lịch trình cố định việc tự động thực hiện Stateless Training**

**Giai đoạn 3: Cố định lịch trình cho việc Statefull Training**

**Giai đoạn 4: Continual Learning**

## **Test In Production**

Một số chiến thuật Test in Production:

### **Shadow Deployment**

Shadow Deployment là chiến thuật deploy một “challenger model” song song với model đã được deploy trước đó. Gửi mọi yêu cầu mới đến cả 2 model, tuy nhiên chỉ trả về cho khách hàng kết quả của model nào có hiệu suất cao hơn. Lưu dự đoán của cả hai models lại để phục vụ cho việc so sánh hiệu suất.

Thuận lợi của Shadow Deployment:

* Đây là cách an toàn nhất để deploy model.
* Chiến thuật này đơn giản về mặt khái niệm.
* Thử nghiệm của bạn sẽ thu thập đủ dữ liệu để đạt được ý nghĩa thống kê nhanh hơn tất cả các chiến lược khác vì tất cả các mô hình đều nhận được lưu lượng truy cập đầy đủ.

Bất lợi của Shadow Deployment:

* Kỹ thuật này không thể đo lường được hiệu suất của model nếu hiệu suất đó phụ thuộc vào phản ứng của người dùng trước dự đoán của model.
* Kỹ thuật này đòi hỏi nhiều tài nguyên khi khởi chạy bởi vì cả 2 model sẽ cùng dữ đoán do đó lượng tính toán cũng sẽ tăng gấp đôi.

### **A/B Testing**

Triển khai “challenger model” bên cạnh model hiện tại, một lượng yêu cầu từ người dùng sẽ được gửi đến “challenger model”, và dự đoán của “challenger model” sẽ được trả trực tiếp về người dùng. Chúng ta sẽ thống kê hiệu suất của cả 2 model để so sánh xem model nào sẽ mang lại kết quả tốt hơn.

Thuận lợi của A/B Testing:

* Vì dự đoán của “challenger model” sẽ được gửi trực tiếp đến người dùng, do đó chúng ta có thể bao quát mọi trường hợp để đánh giá hiệu suất của model.
* A/B Testing dễ hiểu và được nhiều thư viện hỗ trợ trong quá trình thực hiện.
* A/B Testing tiêu tốn ít tài nguyên khi khởi chạy, bởi vì chỉ có 1 dự đoán được thực hiện trên 1 yêu cầu.

### **Canary Release**

Chiến thuật Canary Release deploy “challenger model” và model được deploy trước đó (tạm gọi là “champion model”) bên cạnh nhau. Ban đầu “challenger model” sẽ không nhận được yêu cầu từ khách hàng.

Từ từ chuyển lưu lượng yêu cầu từ “champion model” sang “challenger model”. Đo lường hiệu suất của “challenger model”, nếu hiệu suất tốt cứ tiếp tục việc chuyển lưu lượng yêu cầu sang “challenger model” cho tới khi nào mọi yêu cầu từ người dùng được chuyển hoàn toàn tới “challenger model”.

Thuận lợi của Canary Release:

* Dễ hiểu và dễ thực hiện.
* So với Shadow Deployment, Canary Release tiêu tốn ít chi phí hơn khi vận hành.

Bất lợi của Canary Release:

* Có thể xảy ra khả năng không khắt khe trong việc xác định sự khác biệt về hiệu suất.

### **Interleaving Experiments**

Trong A/B Testing, người dùng chỉ nhận được dự đoán từ model A hoặc model B. Trong Interleaving, người dùng sẽ nhận được xen kẽ dự đoán từ cả model A và model B. Hiệu suất của model được đánh giá thông qua sự ưu tiên của người dùng đối với model A hay model B.

Thuận lợi của Interleaving Experiments:

* Interleaving mang lại hiệu quả tốt trong các bài toán *recommendations* và với một lượng dữ liệu tương đối nhỏ.
* So với Shadow Deployment, Interleaving có thể bao phủ toàn bộ tiêu chí đánh giá hiệu suất của model.

Bất lợi của Interleaving Experiments:

* Tiêu tốn nhiều chi phí trong quá trình vận hành.
* Thực hiện phức tạp hơn A/B tesing.
* Không thể bao phủ hoàn toàn mọi tác vụ. Ví dụ, hiệu suất của Interleaving trong bài toán *recommendation/ ranking* rất tốt, tuy nhiên đối với dạng bài toán *regression* lại không mang lại kết quả khả quan.

### **Bandits**

Bandits là thuật toán mà trong đó chúng ta theo dỗi hiệu suất của toàn bộ các biến thể của model và đưa ra quyết định có nên đưa toàn bộ yêu cầu từ người dùng đến model có hiệu suất tốt nhất hay là sẽ thử một model khác để có thêm thông tin về model đó.

Thuận lợi của Bandits:

* Bandits cần ít data hơn A/B Testing để xác định model nào có hiệu suất tốt hơn.
* So với A/B Tesing, Bandits sẽ là một lựa chọn an toàn hơn. Bởi vì model nào có hiệu suất thấp sẽ ít bị lựa chọn hơn.

Bất lợi của Bandits:

* So với toàn bộ chiến thuật ở trên, Bandits khó thực hiện hơn.
* Bandits chỉ có thể áp dụng trong một số trường hợp nhất định.