TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM
TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG
KHOA CÔNG NGHỆ THÔNG TIN

Lê Đào Duy Tân - 52100104

BÁO CÁO CUỐI KÌ
MÔN NHẬP MÔN HỌC MÁY

THÀNH PHỐ HỒ CHÍ MINH, NĂM 2023

TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM
TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG
KHOA CÔNG NGHỆ THÔNG TIN

Lê Đào Duy Tân - 52100104

BÁO CÁO CUỐI KÌ
MÔN NHẬP MÔN HỌC MÁY

Người hướng dẫn
Ts Lê Anh Cường

THÀNH PHỐ HỒ CHÍ MINH, NĂM 2023

LỜI CẢM ƠN

Lời đầu tiên, em xin chân thành cảm ơn thầy Lê Anh Cường. Trong suốt quá trình học tập môn Nhập môn học máy, thầy đã tận tình hướng dẫn và hỗ trợ cho em được nắm vững các vấn đề cần thiết trong môn này. Hơn hết, thầy đã trang bị đủ cho em những kiến thức một cách đầy đủ để có thể hoàn thành bài báo cáo này.

Tiếp theo, em xin gửi lời cảm ơn sâu sắc tới khoa Công Nghệ Thông Tin trường Đại học Tôn Đức Thắng. Khoa đã tạo mọi điều kiện cho chúng em được học tập và nghiên cứu môn học này. Và đặc biệt các thầy cô trong khoa luôn sẵn sàng chia sẻ những kiến thức bổ ích giúp cho việc thực hiện bài báo cáo của em được hoàn thành một cách tốt nhất.

Cuối cùng, do giới hạn về mặt kiến thức, em biết bài báo cáo của mình còn nhiều thiếu sót và hạn chế, kính mong được sự hướng dẫn và đóng góp của quý thầy cô để bài báo cáo của em được hoàn thiện hơn. Chúc quý thầy cô tràn đầy sức khỏe.

TP. Hồ Chí Minh, ngày 10 tháng 12 năm 2023

Tác giả

Lê Đào Duy Tân

CÔNG TRÌNH ĐƯỢC HOÀN THÀNH
TẠI TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG

Tôi xin cam đoan đây là công trình nghiên cứu của riêng tôi và được sự hướng dẫn khoa học của Ts. Lê Anh Cường. Các nội dung nghiên cứu, kết quả trong đề tài này là trung thực và chưa công bố dưới bất kỳ hình thức nào trước đây. Những số liệu trong các bảng biểu phục vụ cho việc phân tích, nhận xét, đánh giá được chính tác giả thu thập từ các nguồn khác nhau có ghi rõ trong phần tài liệu tham khảo.

Ngoài ra, trong Dự án còn sử dụng một số nhận xét, đánh giá cũng như số liệu của các tác giả khác, cơ quan tổ chức khác đều có trích dẫn và chú thích nguồn gốc.

Nếu phát hiện có bất kỳ sự gian lận nào tôi xin hoàn toàn chịu trách nhiệm về nội dung Dự án của mình. Trường Đại học Tôn Đức Thắng không liên quan đến những vi phạm tác quyền, bản quyền do tôi gây ra trong quá trình thực hiện (nếu có).

TP. Hồ Chí Minh, ngày 10 tháng 12 năm 2023

Tác giả

Lê Đào Duy Tân

BÁO CÁO CUỐI KÌ MÔN NHẬP MÔN HỌC MÁY

TÓM TẮT

Bài bào cáo cá nhân môn Nhập môn học máy gồm 2 chương:

Chương một đề cập đến các phương pháp optimizer như: Gradient descent, Stochastic gradient descent, adam, adargad…

Chương hai nghiên cứu về continual learning và test production trong học máy, đồng thời minh họa bài toán.

MỤC LỤC

DANH MỤC HÌNH VẼ

DANH MỤC CÁC CHỮ VIẾT TẮT

SGD	STOCHASTIC GRADIENT DESCENT
GD	GRADIENT DESCENT
RMSProp	Root Mean Square Propogation

CHƯƠNG 1. CÁC PHƯƠNG PHÁP OPTIMIZER

1.1 Tổng quan

Optimizer là một thuật toán được sử dụng để cập nhật các tham số của mô hình học máy trong quá trình huấn luyện. Optimizer tìm cách tìm ra tập hợp tham số tối ưu nhất để giảm thiểu hàm mất mát của mô hình. Ngoài ra, Optimizer là cơ sở để xây dựng mô hình neral network.

1.2 Các phương pháp optimizer phổ biến

1.2.1 *Gradient descent*

- Vấn đề được đặt ra là trong các bài toán tối ưu là tìm điểm nhỏ nhất của một hàm số. Đánh giá cho thấy như việc tìm global minium của các hàm mất mát rất phức tạp trong machine learning, và có thể là bất khả thi. Tuy nhiên, local minium là nghiệm của phương trình đạo hàm bằng 0 có thể dùng để thay vào các điểm làm cho hàm có giá trị nhỏ nhất, nhưng việc tìm đạo hàm bằng 0 là bất khả thi.
- Gradient Descent là hướng tiếp cận được dùng nhiều nhất để thực hiện việc xuất phát từ một điểm gần với nghiệm của bài toán, sau đó dùng một phép lặp để tiến dần đến điểm cần tìm, nghĩa là đến gần đạo hàm bằng 0.
- Với Gradient Descent cho hàm 1 biến :
  - Xét hàm một biến f(x). Thuật toán sẽ tìm cực tiểu của hàm số bằng cách tạo ra giá trị ngẫu nhiên x. Sau đó, di chuyển x ngược hướng với đạo hàm của f(x) và lặp đi lặp lại cho đến khi đạt tới một ngưỡng nào đó.
  - Công thức cập nhật x như sau :

    $$x_{t+1} = x_t - \alpha . \mathrm{f}'(x_t)$$

    Trong đó :

    x_t là giá trị x tại bước thứ t

    α là learning rate (tốc độ học)

$f'(x_t)$ là đạo hàm của hàm f tại x_t

- Với hàm nhiều biến $f(\vec{x})$:
  - Tính gradient (vector đạo hàm) của hàm f tại một biến ngẫu nhiên $\vec{x}$, sau đó di chuyển $\vec{x}$ ngược hướng với gradient
  - Công thức cập nhật $\vec{x}$:

$$\vec{x}_{t+1} = x_t - \alpha * \nabla f(\vec{x}_t)$$

Trong đó :

$\vec{x}_t$ là giá trị x tại bước thứ t

α là learning rate (tốc độ học)

$f'(\vec{x}_t)$ là đạo hàm của hàm f tại $\vec{x}_t$

- Ưu điểm:
  - Là một phương pháp đơn giản và hiệu quả.
  - Được ứng dụng nhiều trong các mô hình học máy.
- Nhược điểm:
  - Có thể bị quá đà nếu learning rate quá lớn.
  - Có thể không hội tụ nếu hàm mục tiêu có gradient không ổn định.

1.2.2 *Stochastic gradient descent (SGD)*

- Thực hiện tính toán đạo hàm của hàm mất mát dựa trên chỉ một điểm x_i. Sau đó, cập nhật θ trên đọa hàm này. Thực hiện việc này trên từng điểm của bộ dữ liệu sau dó lặp lại quá trình trên.
- Mỗi lần đi qua toàn bộ tập dữ liệu được gọi là một epoch. Trong thuật toán Gradient Descent (GD), thông thường, mỗi epoch tương ứng với việc cập nhật một lần tham số θ. Trong khi đó, với Stochastic Gradient Descent (SGD), mỗi epoch đều đồng nghĩa với N lần cập nhật θ (với N là số điểm dữ liệu).
- Việc cập nhật từng điểm dữ liệu có thể làm giảm tốc độ thực hiện mỗi epoch. Tuy nhiên, từ một góc độ khác, SGD chỉ đòi hỏi một số lượng epoch rất nhỏ (thường chỉ cần khoảng 10 ở lần đầu tiên, sau đó chỉ cần

chạy dưới một epoch là đã có kết quả tốt). Do đó, SGD thích hợp cho các bài toán có cơ sở dữ liệu lớn và các bài toán yêu cầu mô hình thay đổi liên tục.

- Sau mỗi epoch, thứ dự dữ liệu cần được xáo trộn ngẫu nhiên. Quy tác cập nhật của SGD:

$$\theta = \theta - \mathrm{n}\nabla_\theta J(\theta; x_i; y_i)$$

Trong đó:

$J(\theta; x_i; y_i)$: là hàm mất mát với 1 cặp điểm dữ liệu là $(x_i; y_i)$

1.2.3 *Mini-batch gradient descent*

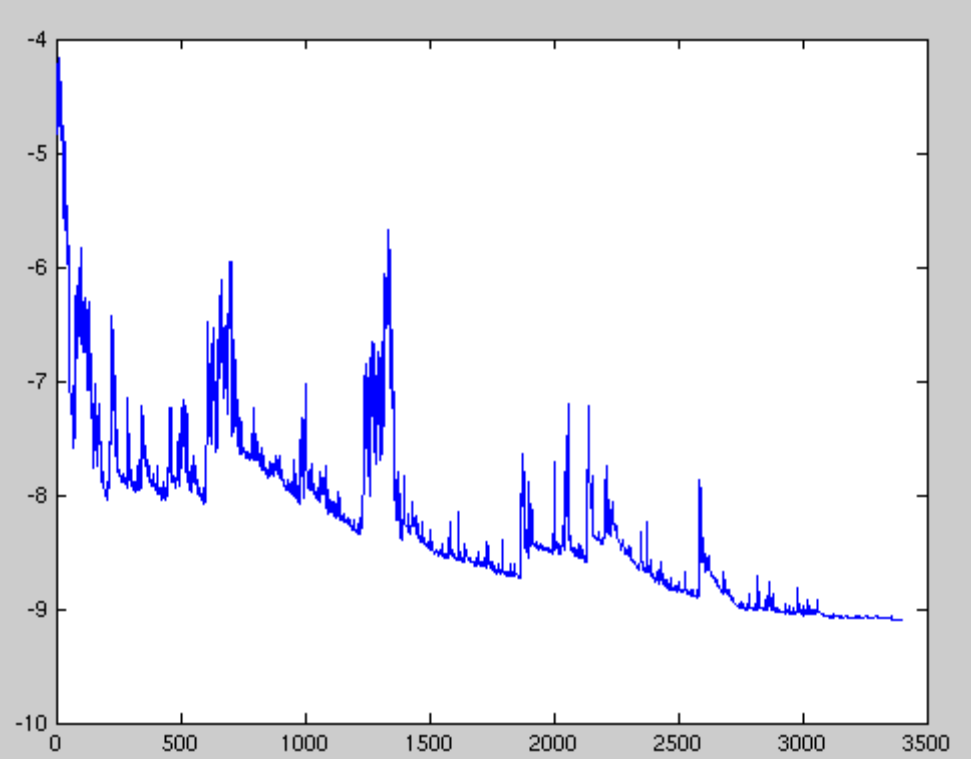
- Sử dụng một số lượng n lớn hơn 1 nhưng nhỏ hơn N (N là số lượng dữ liệu trong bộ dữ liệu). Mini-batch gradient descent bắt đầu mỗi epoch bằng việc đảo lộn ngẫu nhiên dữ liệu và sau đó chia tập dữ liệu thành các mini-batch. Với mỗi mini-batch có n điểm dữ liệu (nếu N không chia hết cho n thì mini batch cuối có thể ít hơn). Sau mỗi lần cập nhật thì lấy ra một mini-batch để tính đạo hàm rồi cập nhật.
- Công thức:

$$\theta = \theta - \mathrm{n}\nabla_\theta J(\theta; x_{i:i+n}; y_{i:i+n})$$

Trong đó:

$x_{i:i+n}$ là dữ liệu thứ i Itới i + n – 1. Dữ liệu này sau mỗi epoch là khác nhau vì chúng cần được xáo trộn.

- Giá trị n thường được chọn là khoảng từ 50 đến 100.



Hình 1.2.1: Hình ảnh hàm mất mát sau mỗi lần cập nhật tham số
(nguồn wikipedia)

- Hàm này lên xuống sau mỗi lần cập nhật nhưng giảm dần và có xu hướng hội tụ ở các điểm cuối.

1.2.4 *SGD with momentum*

- Là phương pháp nhằm tăng tốc độ các vector độ dốc theo đúng hướng. Từ đó, hệ thống sẽ hội tụ nhanh hơn.
- Tình lượng thay đổi tại thời điểm t để cập nhậy vị trí cho nghiệm

$$v_t = \gamma v_{t-1} + n\nabla_\theta J$$

Trong đó:

γ có giá trị khoảng 0.9

v_t là vận tốc tại thời điểm trước đó

$\nabla_\theta J$ là độ dốc của thời điểm trước đó

1.2.5 *RMSProp (Root Mean Square Propogation)*

- Ứng dụng trung bình bình phương của gradient để chuẩn hóa nó nhằm cân bằng kích thước – giảm bước cho độ lớn để tránh hiện tượng Exploding Gradient, và tăng bước cho độ dốc nhỏ để hạn chế Vanishing Gradient.
- RMSProp sẽ điều chỉnh tốc độ học một cách tự động, thêm vào đó nó còn chọn một tỉ lệ học tập cho mỗi tham số một cách khác nhau.
- Công thức cập nhật trọng số:

$$s_t = \rho s_{t-1} + (1 - \rho).g_t^2$$

$$\Delta x_t = -\frac{n}{\sqrt{s_t + \in}}.g_t$$

$$x_{x+1} = x_t + \Delta x_t$$

Trong đó:

s_t là tích lũy phương sai của các gradient trong quá khứ

ρ là tham số suy giảm

Δx_t là sự thay đổi tham số

g_t là gradient của các tham số tại vòng lặp t

$\in$ là tham số đảm bảo kết quả xấp xỉ có ý nghĩa

1.2.6 *Adagrad*

- Kỹ thuật Adagrad tiến hành giảm dần tốc độ thông qua việc thay đổi tốc độ họ. Adagrad được cải thiện hơn thông qua việc cho trọng số học chính xác dựa vào đầu vào trước nó nhằm tự điều chỉnh tỉ lệ học theo cách tối ưu nhất.
- Công thức :

$$w_{t+1} = w_t - \frac{n}{\sqrt{G_t + \in}} . g_t$$

$Trong$ đó :

G_t là ma trận đường chéo chứa bình phương của đạo hàm vector tham số tại vòng lặp t

g_t là vector của độ dốc cho vị trí hiện tại

n là tỉ lệ học

1.2.7 *Adadelta*

- Đối với Adadelta không có tham số tỉ lệ học. Với Adadelta sử dụng tốc độ thay đổi của chính các tham số để điều chỉnh tỉ lệ học (giới hạn cửa sổ của gradient tích lũy trong quá khứ ở một số kích thước cố định của trọng số).
- Công thức :

$$g'_t = \sqrt{\frac{\Delta x_{t-1} + \epsilon}{s_t + \epsilon}} . g_t$$

$$x_t = x_{t-1} - g'_t$$

$$\Delta x_t = \rho \Delta x_{t-1} + (1 - \rho) x_t^2$$

Hình 1.2.2: Hình ảnh công thức Adadelta

- Trong đó:

  s_t là để lưu trữ trung bình của khoảng thời gian thứ hai của gradient

  Δx_t dùng để trung bình của khoảng thời gian thứ 2 của sự thay đổi các tham số

  g'_t là căn bậc hai thương của trung bình tốc độ thay đổi bình phương và trung bình mô-men bậc hai của gradient.

1.2.8 *Adam*

- Tính toán tỉ lệ học cá nhân cho các tham số khác nhau. Nó ứng dụng ước tính của khoảng thời gian thứ nhất và thứ hai của độ dốc để điều chỉnh tỉ lệ học cho từng trọng số của mạng nơ-ron.
- Công thức:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$
$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

Hình 1.2.3: Hình ảnh mô tả công thức của Adam

- Trong đó:

  V_t là trung bình động của bình phương

  m_t là trung bình động của gradient

  β_1 và β_2 là tốc độ của di chuyển

1.3 So sánh

- Dataset : Mnits

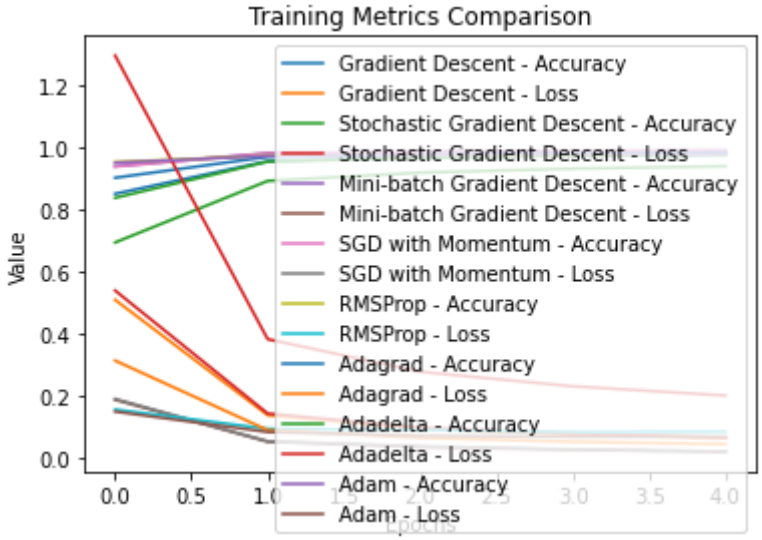
```
import tensorflow as tf
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Conv2D, MaxPooling2D
from tensorflow.keras.optimizers import SGD, Adam, RMSprop, Adagrad, Adadelta
import matplotlib.pyplot as plt

# Define hyperparameters
batch_size = 32
epochs = 5
learning_rate = 0.01

# Load MNIST data
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# Normalize data
x_train = x_train.astype("float32") / 255.0
x_test = x_test.astype("float32") / 255.0
```

```
    # Reshape data for CNN
    x_train = x_train.reshape(x_train.shape[0], 28, 28, 1)
    x_test = x_test.reshape(x_test.shape[0], 28, 28, 1)

    # One-hot encode labels
    y_train = tf.keras.utils.to_categorical(y_train, 10)
    y_test = tf.keras.utils.to_categorical(y_test, 10)

    # Define optimizer dictionaries
    optimizers = {
        "Gradient Descent": SGD(learning_rate=learning_rate),
        "Stochastic Gradient Descent": SGD(learning_rate=learning_rate),
        "Mini-batch Gradient Descent": SGD(learning_rate=learning_rate, momentum=0.9),
        "SGD with Momentum": SGD(learning_rate=learning_rate, momentum=0.9),
        "RMSProp": RMSprop(learning_rate=learning_rate),
        "Adagrad": Adagrad(learning_rate=learning_rate),
        "Adadelta": Adadelta(learning_rate=learning_rate),
        "Adam": Adam(learning_rate=learning_rate),
    }

    # Train and evaluate models with each optimizer
    results = {}
    for name, optimizer in optimizers.items():
        # Create and compile model
        model = Sequential()
        model.add(Conv2D(32, (3, 3), activation="relu", input_shape=(28, 28, 1)))
```

```
            model.add(MaxPooling2D((2, 2)))
            model.add(Conv2D(64, (3, 3), activation="relu"))
            model.add(MaxPooling2D((2, 2)))
            model.add(Flatten())
            model.add(Dense(128, activation="relu"))
            model.add(Dense(10, activation="softmax"))
            model.compile(loss="categorical_crossentropy", optimizer=optimizer, metrics=["accuracy"])

            # Train and store results
            history = model.fit(x_train, y_train, epochs=epochs, batch_size=batch_size)
            results[name] = (history.history["accuracy"], history.history["loss"])

        # Compare results
        for name, (acc, loss) in results.items():
            print(f"{name}:")
            print(f"- Accuracy: {acc[-1]}")
            print(f"- Loss: {loss[-1]}")

        # Plot accuracy and loss curves (optional)
        for name, (acc, loss) in results.items():
            plt.plot(acc, label=f'{name} - Accuracy')
            plt.plot(loss, label=f'{name} - Loss')

        plt.xlabel('Epochs')
        plt.ylabel('Value')
        plt.legend()
        plt.title('Training Metrics Comparison')
```

```
    plt.show()
```

- Biểu đồ từ đoạn code sau :



- Biểu đồ cho thấy rằng Adam là phương pháp đào tạo có độ chính xác cao nhất. Phương pháp này có thể đạt được độ chính xác cao hơn các phương pháp khác trong thời gian ngắn hơn.
- Adam có độ chính xác cao nhất ở tất cả các thời điểm đào tạo. Điều này cho thấy rằng Adam là một phương pháp đào tạo hiệu quả, có thể đạt được độ chính xác cao ngay cả khi có ít dữ liệu đào tạo.
- Các phương pháp khác có độ chính xác thấp hơn Adam. Gradient Descent (GD) có độ chính xác thấp nhất, tiếp theo là Stochastic Gradient Descent (SGD). Các phương pháp khác có độ chính xác tương đương nhau, nhưng không cao bằng Adam.
- Các phương pháp khác có thời gian hội tụ chậm hơn Adam. GD và SGD có thể mất nhiều thời gian để hội tụ, đặc biệt là khi sử dụng các hàm mất mát không lồi. Các phương pháp khác có thời gian hội tụ ngắn hơn, nhưng vẫn chậm hơn Adam.

CHƯƠNG 2. CONTINUAL LEARNING VÀ TEST PRODUCTION

2.1 Tổng quan về continual learning

- Continual learning là khi có dữ liệu mới thì mô hình sẽ được cập nhật nhằm giúp mô hình duy trì phân phối dữ liệu hiện tại.
- Continual learning đáp ứng việc thay đổi trong phân phối dữ liệu.
- Mục tiêu là giữ cho mô hình hiệu suất cao trên tất cả các dạng dữ liệu, thậm chí khi có sự biến động trong phân phối của chúng.

2.2 Mục đích

- Mục đích cơ bản là để theo kịp sự thay đổi phân phối dữ liệu.
- Tăng cường hiệu suất cho mô hình học máy, có thể được cập nhật để cải thiện hiệu suất của chúng theo thời gian, ví dụ như bằng cách học các mẫu mới hoặc cập nhật các thông số của mô hình. Điều này giúp mô hình học máy có thể thích ứng với các thay đổi trong dữ liệu hoặc môi trường, từ đó cải thiện độ chính xác của các dự đoán.
- Tăng cường khả năng thích ứng cho mô hình học máy để có thể được cập nhật để thích ứng với các thay đổi trong dữ liệu hoặc môi trường. Điều này giúp mô hình học máy có thể đáp ứng tốt hơn với các nhu cầu của khách hàng.
- Giảm chi phí: mô hình học máy có thể được cập nhật mà không cần phải xây dựng lại mô hình từ đầu. Với điều này giúp tiết kiệm chi phí và thời gian cho việc phát triển và triển khai các mô hình học máy.

2.3 Thách thức trong continual learning

- Mô hình có thể quên kiến thức cũ khi học học một kiến thức mới, điều này gọi là catastrophic Forgetting.

- Dữ liệu mới có thể có các đặc trưng, phân phối khác nhau dẫn đến việc mô hình phải thích ứng một cách linh hoạt.

2.4 Giải pháp

- Sử dụng kỹ thuật như Elastic Weight Consolidation (EWC) nhằm giữ lại các trọng số quan trọng cho nhiệm vụ trước đó.
- Bằng cách tích hợp bộ nhớ để lưu trữ thông tin quan trọng từ các nhiệm vụ trước.
- Học từng phần nhỏ của dữ liệu mới một cách tuần tự và kết hợp chúng với kiến thức cũ.

2.5 Tổng quan về test production

- Đây là giai đoạn trong quy trình phát triển mô hình học máy sau khi đã được đào tạo và chuẩn bị triển khia trong môi trường thực tế.
- Giai đoạn này mô hình được kiểm thử chặt chẽ để đảm báo tính ổn định và hiệu suất.

2.6 Hoạt động

- Quanlity Assurance giúp cho mô hình hoạt động đúng đắn và hiệu suất cao khi có dữ liệu mới.
- Performance Testing giúp đánh giá khả năng xử lý và thời gian đáp ứng trong các điều kiện khác nhau.
- Reliability Testing đảm bảo cho mô hình ổn định và đang tin cậy trong thực tế.

2.7 Công cụ và kĩ thuật

- A/B testing là so sánh hiệu suất giữa mô hình mới và mô hình cũ trong sản xuất.
- Cross-Validation giúp đánh giá hiệu suất trên nhiều tập dữ liệu nhằm bảo đảm tính chất tổng quát.

- Monitoring và Logging để theo dõi hiệu suất của mô hình trong thời gian thực và khắc phục sự cố nhanh chóng.

2.8 Mô phỏng

```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.datasets import load_iris
import numpy as np

iris = load_iris()
X, y = iris.data, iris.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

initial_model = RandomForestClassifier(n_estimators=100, random_state=42)
initial_model.fit(X_train, y_train)

y_pred_initial = initial_model.predict(X_test)
accuracy_initial = accuracy_score(y_test, y_pred_initial)
print(f"Initial Test Accuracy: {accuracy_initial}")

X_new = X + np.random.normal(scale=0.1, size=X.shape)
y_new = y

continual_model = initial_model
continual_model.fit(X_new, y_new)
```

```
X_production = X + np.random.normal(loc=0.5, scale=0.2, size=X.shape)
y_production = y

y_pred_production = continual_model.predict(X_production)
accuracy_production = accuracy_score(y_production, y_pred_production)
print(f"Production Test Accuracy: {accuracy_production}")
```

- Kết quả đầu ra:

```
Initial Test Accuracy: 1.0
Production Test Accuracy: 0.7733333333333333
```

- Giải thích: kết quả cho thấy rằng mô hình hoạt động tốt trên dữ liệu kiểm thử ban đầu với độ chính xác là 1.0. Tuy nhiên, khi kiểm thử mô hình trên dữ liệu (production), độ chính xác giảm xuống còn 0.7733.

TÀI LIỆU THAM KHẢO

Tiếng Việt

[1] https://machinelearningcoban.com/2017/01/12/gradientdescent/

[2]https://jos.husc.edu.vn/backup/upload/vol_18/no_1/668_fulltext_4.%C4%90TVT%20-%20Phuoc%20-%20Vuong%20Quang%20Phuoc.pdf

Tiếng anh

[2]Trevor Hastie, Robert Tibshirani, Jerome H. Friedman, [2017], The Elements of Statistical Learning - Data Mining, Inference and Prediction - 2nd Edition, Springer