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

HUỲNH QUỐC TRƯỜNG - 52101007

TÌM HIỂU OPTIMIZER, CONTINUAL LEARNING VÀ TEST PRODUCTION KHI XÂY DỰNG MÔ HÌNH HỌC MÁY

BÁO CÁO CUỐI KÌ
NHẬP MÔN HỌC MÁY

Ho Chi Minh City, 2023.

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

HUỲNH QUỐC TRƯỜNG - 52101007

TÌM HIỂU OPTIMIZER, CONTINUAL LEARNING VÀ TEST PRODUCTION KHI XÂY DỰNG MÔ HÌNH HỌC MÁY

BÁO CÁO CUỐI KÌ
NHẬP MÔN HỌC MÁY

Ho Chi Minh City, 2023.

Mục lục

Danh sách hình vẽ

Danh sách bảng

Danh sách từ viết tắt

Adam Adaptive Moment Estimation

CL Continual Learning

GD Gradient Descent

ML Machine Learning

RMS Root Mean Square

SGD Stochastic Gradient Descent

TP Test Production

Chương 1

GIỚI THIỆU

Trong cuộc sống hiện nay, học máy (machine learning) là một phần không thể thiếu trong các việc giải quyết các vấn đề mà con người có thể không thể giải quyết được trong nhiều lĩnh vực khác nhau. Việc xây dựng một mô hình hiệu quả và tối ưu là một phần rất quan trọng, đòi hỏi việc tối ưu hóa rất cao trong việc điều chỉnh các tham số để đạt được hiệu suất tốt nhất

Tính quan trọng của tối ưu hóa mô hình huấn luyện trong học máy bao gồm:

1. Hiệu quả và độ chính xác
2. Tiết kiệm tài nguyên
3. Khả năng di động

Tính ứng dụng của Continual Learning trong Xây Dựng Giải Pháp Máy Học.

Continual learning, hay còn được gọi là học liên tục, đặt ra nhiều thách thức về việc duy trì và nâng cao kiến thức học được của mô hình huấn luyện khi có sự thay đổi trong dữ liệu hoặc môi trường học. Điều này đặt ra nhu cầu xây dựng các giải pháp máy học có khả năng học và thích ứng liên tục.

1. Duy trì kiến thức học được
2. Khả năng thích ứng và mở rộng
3. Khả năng ứng dụng linh hoạt trên nhiều nền tảng

Chương 2

TÌM HIỂU VÀ SO SÁNH OPTIMIZER

2.1 Tổng quan về Optimizer

2.1.1 Khái niệm

Optimizer là các thuật toán điều chỉnh các tham số của mô hình trong quá trình đào tạo để giảm thiểu hàm mất mát. Chúng cho phép mạng lưới thần kinh học hỏi từ dữ liệu bằng cách cập nhật lặp đi lặp lại các trọng số và độ lệch. Ngoài ra, Optimizer xác định cách mà trọng số của mô hình được cập nhật dựa trên gradient của hàm mất mát. Các thuật toán tối ưu phổ biến bao gồm như:

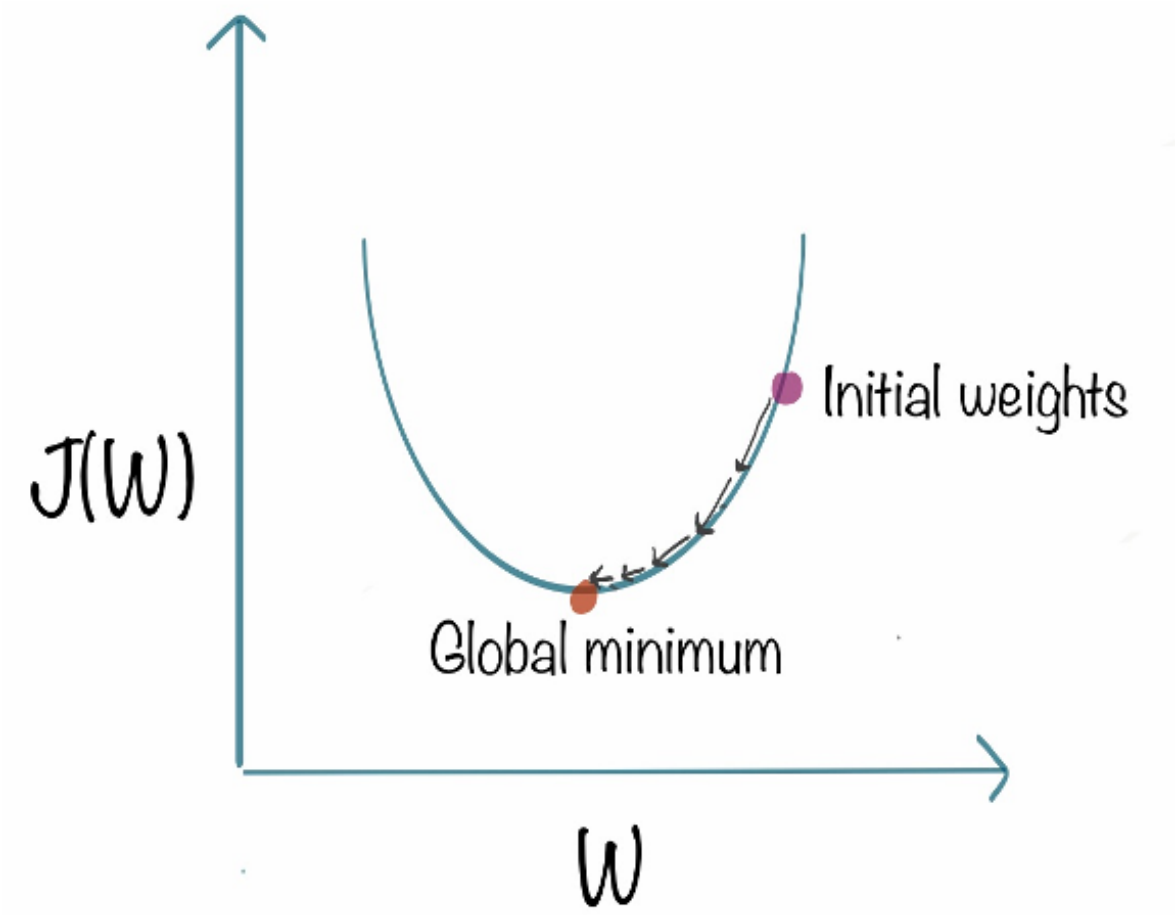
- Gradient Descent (GD)
- Stochastic Gradient Descent (SGD)
- RMSprop
- Adam

2.1.2 Các loại thuật toán tối ưu

Gradient Desent (GD)

Nói một cách đơn giản, Gradient Desent là trong những thuật toán tối ưu dùng để tìm ra giá trị cục bộ tối thiểu của hàm khả vi. Ngoài ra còn được sử dụng để tìm ra các giá trị tham số nhằm giảm thiểu hàm chi phí đến mức tối thiểu.

Hiểu nôm na, độ dóc đơn giản là độ đo lường sự thay đổi các trọng số của sai số. Ta có thể xem GD là độ dốc của hàm, độ dốc càng cao và càng dốc thì mô hình học càng nhanh. Nhưng nếu như độ dốc bằng 0, thuật toán sẽ dừng lại.



Hình 2.1: Minh họa thuật toán Gradient Desent

Các bước hoạt động

1. Khởi tạo giá trị ban đầu cho các tham số
2. Dự đoán
    - Sử dụng các trọng số và độ lệch để dự đoán đầu ra
3. Tính toán đạo hàm
    - Tính toán đạo hàm của hàm chi phí theo các trọng số và độ lệch
4. Cập nhật các trọng số và độ lệch

- Cập nhật các trọng số và độ lệch theo hướng ngược đạo hàm nhằm mục đích giảm thiểu hàm mất mát bằng cách sử dụng công thức Gradient Desent
- Công thức cập nhật Gradient Desent: $w_{i+1} = w_i - \gamma \frac{\partial L}{\partial w_i}$
  - w_{i+1}: trọng số mới
  - w_i: trọng số cũ
  - γ: hệ số chờ
  - $\frac{\partial L}{\partial w_i}$: đạo hàm của hàm chi phí theo trọng số

5. Tiến trình sẽ được lập lại cho đến khi đạt được điều kiện dừng (điều kiện dừng có thể là số lần lặp được tối ưu hoặc khi giá trị hàm mất mát đủ nhỏ)

Ưu điểm

- Đơn giản và dễ hiểu
- Dễ cài đặt
- Hiệu quả với các bài toán có số chiều nhỏ
- Dễ dàng tìm được điểm cực tiểu toàn cục

Nhược điểm

- Với các bài toán có số chiều lớn, GD có thể gặp khó khăn trong việc tìm được điểm cực tiểu toàn cục
- Cần phải chọn hệ số học tập phù hợp
- Cần phải chọn điểm khởi tạo phù hợp

Stochastic Gradient Descent (SGD)

Như đã nói ở trên, GD có thể gặp khó khăn trong việc tìm được điểm cực tiểu toàn cục. Để khắc phục điều này, SGD được ra đời. Stochastic Gradient Descent (SGD) là một phương pháp tối ưu hóa được sử dụng để tìm ra giá trị tối ưu của một hàm mất mát. Nó là một phương pháp tối ưu hóa đơn giản và hiệu quả cho các mô hình học máy bằng cách thay vì lấy toàn bộ tập dữ liệu để tính gradient, SGD chỉ lấy một mẫu ngẫu nhiên từ tập dữ liệu để tính gradient.

Công thức Stochastic Gradient Descent

$$w = w - \eta \nabla Q_i(w)$$

Các bước hoạt động

1. Khởi tạo giá trị ban đầu cho các tham số w và hệ số học tập η
2. Xáo trộn dữ liệu mỗi lần lặp cho đến khi đạt được điều kiện dừng

Ưu điểm

- Hiệu quả với các bài toán có số chiều lớn
- Dễ dàng tìm được điểm cực tiểu toàn cục

Nhược điểm

- Do SGD không sử dụng toàn bộ dữ liệu để tính gradient mà chỉ sử dụng một mẫu ngẫu nhiên nhỏ nên đường đi của nó sẽ không ổn định và dao động quanh điểm cực tiểu. Vì thế SGD có số lần lặp cao hơn nhiều so với GD. Do số lần lặp tăng lên có thể dẫn đến thời gian huận luyện cho tổng thể tăng theo và có thể dẫn đến việc tăng chi phí tính toán.
- SGD có thể không tìm được điểm cực tiểu toàn cục do đường đi của nó dao động quanh điểm cực tiểu.

RMSprop (Root Mean Square Propagation)

Trước khi đi đến RMSprop, chúng ta cùng tìm hiểu về thuật toán Adagrad.

Khái niệm Adagrad

Adagrad được viết tắt từ Adaptive Gradient Descent là một thuật toán tối ưu hóa gradient dựa trên gradient của hàm mất mát được sử dụng cho việc huấn luyện các mạng nơ-ron học sâu. Adagrad được thiết kế để cập nhật trọng số của mạng nơ-ron trong quá trình huấn luyện bằng cách thực hiện một tốc độ học tập riêng biệt cho mỗi tham số được biên thiên theo tham số t. Các tham số càng thay đổi thì tốc độ học tập càng giảm.

Công thức Adagrad

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \odot g_t$$

Trong đó:

- θ_t: tham số tại thời điểm t
- η: tốc độ học tập

- g_t: gradient tại thời điểm t
- G_t: 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.
- ϵ: một số rất nhỏ để tránh chia cho 0
- $\odot$: phép nhân Hadamard
- $G_t = G_{t-1} + g_t^2$

Nhược điểm của Adagrad: độ lớn của tổng bình phương biến thiên G_t sẽ ngày càng tăng lên cho đến khi nó trở nên rất lớn và tốc độ học tập sẽ giảm dần về 0. Điều này có nghĩa là thuật toán sẽ không thể hội tụ đến điểm cực tiểu.

Khái niệm RMSprop

RMSprop là một phương pháp tối ưu hóa gradient dựa trên gradient của hàm mất mát được sử dụng cho việc huấn luyện các mạng nơ-ron học sâu. RMSprop là một phần mở rộng của thuật toán Adagrad. RMSprop được thiết kế để cập nhật trọng số của mạng nơ-ron trong quá trình huấn luyện bằng cách thực hiện một tốc độ học tập riêng biệt cho mỗi tham số được biên thiên theo tham số t. Để giải quyết vấn đề của Adagrad, RMSprop sử dụng bằng cách chia tỷ lệ tốc độ học tập cho một trung bình bình phương của gradient.

Công thức RMSprop

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} \odot g_t$$

Trong đó:

- θ_t: tham số tại thời điểm t
- η: tốc độ học tập
- g_t: gradient tại thời điểm t
- $E[g^2]_t$: trung bình bình phương của gradient
- ϵ: một số rất nhỏ để tránh chia cho 0
- $\odot$: phép nhân Hadamard
- $E[g^2]_t = \gamma E[g^2]_{t-1} + (1-\gamma) g_t^2$

- γ: hệ số chờ

Ưu điểm của RMSprop

- Giúp thuật toán hội tụ nhanh hơn
- Giúp giảm bớt vấn đề của Adagrad khi độ lớn của tổng bình phương biến thiên G_t ngày càng tăng lên
- Giúp tốc độ học tập không bị giảm quá nhanh
- Giúp thuật toán có thể hội tụ đến điểm cực tiểu

Nhược điểm của RMSprop

- RMSprop có thể tìm ra kết quả là local minimum (điểm cực tiểu cục bộ) thay vì global minimum (điểm cực tiểu toàn cục) do xuất phát tự sự thích ứng của learning rate η vì giúp cho việc hội tụ nhanh hơn tại các điểm critical của hàm mất mát, tuy nhiên nó cũng có thể dẫn đến việc bị bẻ cong tại các điểm local minimum khiến cho mô hình không thể hội tụ đến global minimum. Ngoài ra, cũng có thể do khả năng dừng lại tại các điểm local minimum của thuật toán.

Adam

Adam là từ viết tắt của *Adaptive Moment Estimation*(ước tính thời điểm thích ứng). Adam là một thuật toán tối ưu hóa gradient dựa trên gradient của hàm mất mát được sử dụng cho việc huấn luyện các mạng nơ-ron học sâu. Adam là một phần mở rộng của thuật toán SGD và được thiết kế để cập nhật trọng số của mạng nơ-ron trong quá trình huấn luyện. Không giống như SGD cố gắng duy trì cùng một tốc độ học tập cho tất cả các trọng số, Adam sử dụng các tốc độ học tập riêng biệt cho từng tham số. Nó cũng tính toán các tốc độ học tập được điều chỉnh cho từng tham số dựa trên độ dốc trong quá khứ và độ dốc hiện tại của chúng. Adam là sự kết hợp của RMSprop và thuật toán Momentum.

*** Momentum**

Nói tóm tắt một chút về Momentum (Quán tính), Momentum là một phương pháp tối ưu hóa gradient dựa trên gradient của hàm mất mát được sử dụng cho việc huấn luyện các mạng nơ-ron học sâu. Momentum là một phần mở rộng của thuật toán SGD và được thiết kế để cập nhật trọng số của mạng nơ-ron trong quá trình huấn luyện.

Momentum cố gắng giảm bớt sự dao động của gradient bằng cách tính toán trung bình có trọng số của các gradient trước đó.

Công thức Momentum

$$v_t = \gamma v_{t-1} + \eta \nabla Q_i(w)$$

Trong đó:

- v_t: là véc tơ quán tính tại thời điểm t
- γ: hệ số quán tính
- η: tốc độ học tập
- $\nabla Q_i(w)$: gradient tại thời điểm t
- w: tham số
- $v_0 = 0$
- $w = w - v_t$

Công thức Adam

$$w_{t+1} = w_t - \frac{\eta}{\sqrt{\hat{v}_t + \epsilon}} \hat{m}_t$$

Trong đó:

- w_{t+1}: tham số tại thời điểm t+1
- w_t: tham số tại thời điểm t
- η: tốc độ học tập
- $\hat{v}_t$ và $\hat{m}_t$ là các ước lượng được điều chỉnh các động lượng thứ nhất và thứ hai của gradient. Trong đó:
  - $\hat{m}_t = \frac{m_t}{1-\beta_1^t}$
  - $\hat{v}_t = \frac{v_t}{1-\beta_2^t}$
  * $m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$
  * $v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$
- ϵ: một số rất nhỏ để tránh chia cho 0

Ưu điểm của Adam

- Thích ứng với các bài toán có độ lớn của gradient thay đổi theo thời gian
- Sự kết hợp của RMSprop và thuật toán Momentum giúp cân bằng giữa việc theo đuổi định hướng của gradient và việc giảm bớt sự dao động của gradient nhằm tối ưu hóa hàm mất mát trên các bài toán phức tạp

Nhược điểm của Adam

- Tuy Adam thích ứng cao với các bài toán có độ lớn những cũng khá nhạy cảm với các bộ dữ liệu nhiễu, đặc biệt là các trường hợp dữ liệu không đồng nhất. Nếu gradient có độ lớn lớn và biên độ dao động cao, Adam có thể không ổn định và khó khăn khi hội tụ.
- Adam cần được cấu hình đúng để đảm bảo hiệu suất tối ưu. Các tham số như β_1, β_2 và ϵ cần được điều chỉnh phù hợp để đảm bảo Adam hoạt động tốt.
- Tham số dùng cho bài toán này thường nhiều hơn so với các thuật toán tối ưu khác. Điều này có thể làm tăng bộ nhớ và thời gian tính toán cần thiết cho Adam.

2.2 So sánh các thuật toán tối ưu

Để hoàn thành cho việc mục tiêu cho việc so sánh các thuật toán tối ưu với nhau. Ở đây chúng ta sẽ sử dụng các thuật toán với bộ dữ liệu mnist (bộ dữ liệu về chữ số viết tay) với Tensorflow.

```
1  from tensorflow import keras
2  from keras.datasets import mnist
3  from keras.models import Sequential
4  from keras.layers import Dense, Dropout, Flatten
5  from keras.layers import Conv2D, MaxPooling2D
6  import pandas as pd
7  import time
8  import numpy as np
9  import matplotlib.pyplot as plt
10
11 # load mnist data
```

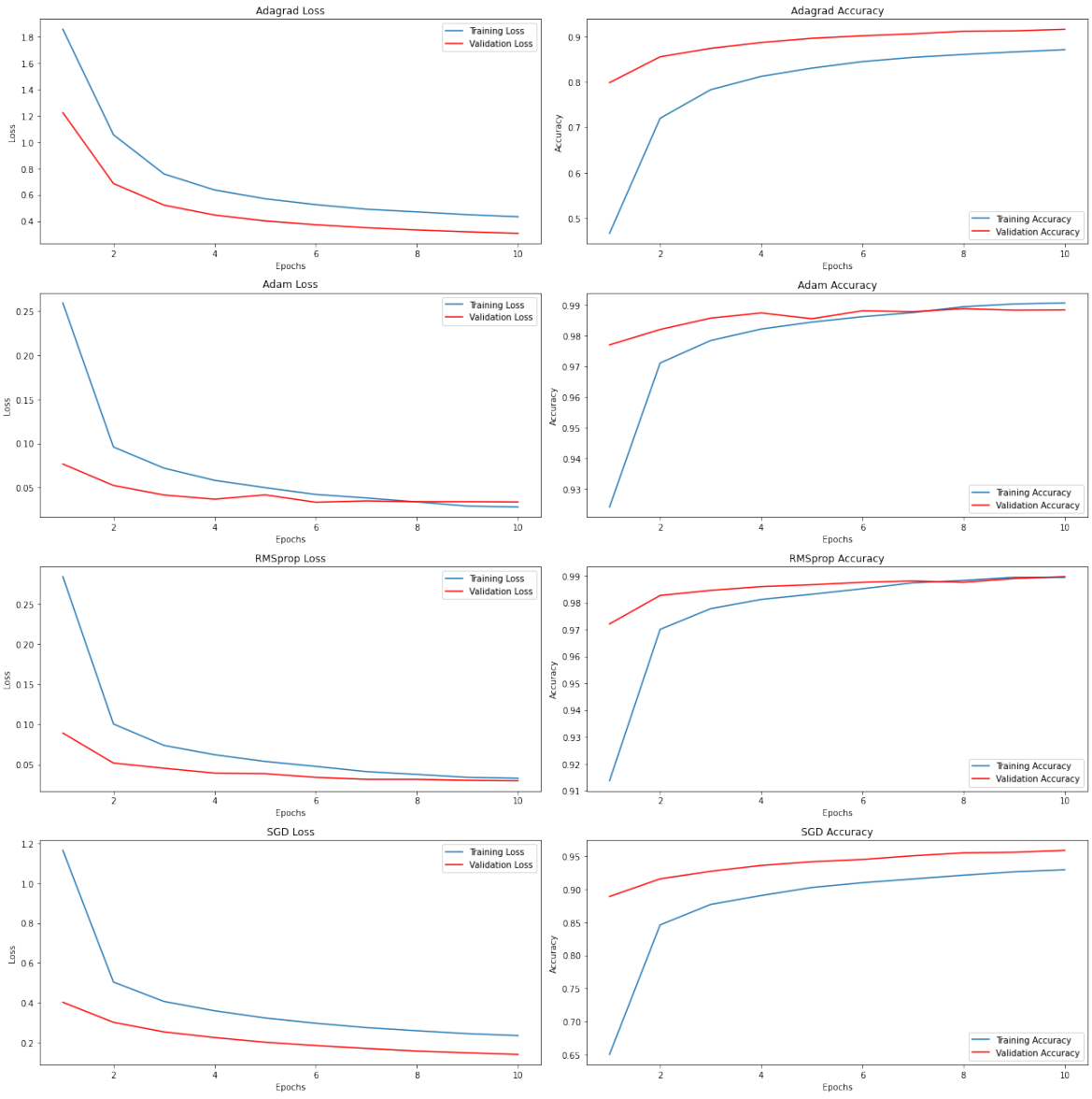
```
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# reshape data
'''
    (60000, 28, 28) <-> mean 60000 images, 28x28 pixels
    (60000, 28, 28, 1) <-> 60000 images, 28x28 pixels, 1 color
    ↪ channel
'''
x_train= x_train.reshape(x_train.shape[0],28,28,1)
x_test=  x_test.reshape(x_test.shape[0],28,28,1)
input_shape=(28,28,1)
y_train=keras.utils.to_categorical(y_train)#,num_classes=)
y_test=keras.utils.to_categorical(y_test)#, num_classes)
x_train= x_train.astype('float32')
x_test= x_test.astype('float32')
x_train /= 255
x_test /=255

batch_size=128
num_classes=10
epochs=10
# create model
def create_model(optimizer):
    model=Sequential()
    model.add(Conv2D(
        32,
        kernel_size=(3,3),
        activation='relu',
        input_shape=input_shape)
    )
    model.add(MaxPooling2D(pool_size=(2,2)))
    model.add(Dropout(0.25))
    model.add(Flatten())
    model.add(Dense(256, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(num_classes, activation='softmax'))
```

```
47          model.compile(loss=keras.losses.categorical_crossentropy,
            ↪  optimizer= optimizer, metrics=['accuracy'])
48          return model
49
50      # create table
51      optimizers = ['Adagrad', 'Adam', 'RMSprop', 'SGD']
52      table = pd.DataFrame(columns=['Optimizer', 'Test loss', 'Test
        ↪  accuracy', 'Epochs', 'Total time'])
53      rows, cols = len(optimizers), 2
54      fig, axes = plt.subplots(rows, cols, figsize=(18,18))
55
56      for index, optimizer in enumerate(optimizers):
57
58          model = create_model(optimizer)
59          start_time = time.time()
60          # fit model
61          hist = model.fit(x_train, y_train, batch_size=batch_size,
            ↪  epochs=epochs, verbose=1, validation_data=(x_test,y_test))
62
63          train_loss, val_loss = hist.history['loss'],
            ↪  hist.history['val_loss']
64          train_acc, val_acc = hist.history['accuracy'],
            ↪  hist.history['val_accuracy']
65          epochs_range = range(1, epochs+1)
66          axes[index][0].plot(epochs_range, train_loss, label='Training
            ↪  Loss')
67          axes[index][0].plot(epochs_range, val_loss, label='Validation
            ↪  Loss', color='red')
68
69          axes[index][1].plot(epochs_range, train_acc, label='Training
            ↪  Accuracy')
70          axes[index][1].plot(epochs_range, val_acc, label='Validation
            ↪  Accuracy', color='red')
71
72          axes[index][0].set_title(optimizer + ' Loss')
73          axes[index][0].set_xlabel('Epochs')
```

```
74          axes[index][0].set_ylabel('Loss')
75          axes[index][0].legend()
76
77          axes[index][1].set_title(optimizer + ' Accuracy')
78          axes[index][1].set_xlabel('Epochs')
79          axes[index][1].set_ylabel('Accuracy')
80          axes[index][1].legend()
81
82          end_time = time.time()
83          # evaluate model
84          score = model.evaluate(x_test, y_test, verbose=0)
85          print('Optimizer: ', optimizer)
86          print('Test loss: ', score[0])
87          print('Test accuracy: ', score[1])
88          total_time = end_time - start_time
89          table.loc[len(table)] = [optimizer, score[0], score[1],
            ↪  epochs, total_time]
90
91      plt.tight_layout()
92      plt.show()
93      print(table)
```



Hình 2.2: Kết quả chạy chương trình

Chương 3

Continual Learning và Test Production

3.1 Continual Learning

3.1.1 Giới thiệu

Theo ngữ cảnh của bài toán, Continual Learning (CL) có thể được hiểu là một hướng tiếp cận trong Machine Learning (ML) mà mô hình được huấn luyện trên một chuỗi các tác vụ (task) liên tục. Mỗi tác vụ có thể là một bài toán khác nhau hoặc là một phần của một bài toán lớn. Mục tiêu của CL là giúp mô hình có thể học được nhiều tác vụ khác nhau mà không quên đi những gì đã học được từ các tác vụ trước đó. Điều này có nghĩa là mô hình phải có khả năng học được các kiến thức có tính chất chung (common knowledge) giữa các tác vụ. Ví dụ, khi mô hình được huấn luyện để phân loại các loại động vật, nó sẽ học được các kiến thức chung như: động vật có thể di chuyển, có thể ăn, có thể sinh sản, có thể thở, có thể phát ra âm thanh, có thể có màu sắc, có thể có hình dạng, có thể có kích thước, có thể có khối lượng, có thể có tuổi. Ngoài ra, CL còn được định nghĩa là một thuật toán thích ứng học tập từ nguồn thông tin liên tục dần có theo thời gian.

3.1.2 Các thách thức của Continual Learning

Catastrophic Forgetting (Quên nặng nề)

Catastrophic Forgetting : Một trong những thách thức chính trong việc CL là ngăn chặn đi tình trạng quên lãng. Khi mô hình được huấn luyện trên một tác vụ mới, nó có thể quên đi những gì đã học được từ các tác vụ trước đó. Điều này dẫn đến việc mô

hình không thể đưa ra được các dự đoán chính xác trên các tác vụ đã được học trước đó. Để giải quyết vấn đề này có thể sử dụng một số phương pháp như: *Regularization methods, Replay buffers,...*

Bộ đệm phát lại (Replay Buffer)

Replay Buffer : Là một phương pháp được sử dụng để giải quyết vấn đề quên lãng. Khi mô hình được huấn luyện trên một tác vụ mới, nó sẽ lưu lại một số dữ liệu của tác vụ đó vào bộ đệm. Sau đó, khi mô hình được huấn luyện trên một tác vụ khác, nó sẽ lấy ra một số dữ liệu từ bộ đệm để huấn luyện cùng với dữ liệu của tác vụ mới. Điều này giúp mô hình không quên đi những gì đã học được từ các tác vụ trước đó. Tuy nhiên, việc lưu trữ dữ liệu vào bộ đệm sẽ tốn không gian lưu trữ và tốn thời gian huấn luyện. Ngoài ra, việc lưu trữ dữ liệu cũng có thể gây ra việc mô hình bị overfitting.

Phương pháp chính quy hóa (Regularization methods)

Regularization methods : Là một phương pháp được sử dụng để giải quyết vấn đề quên lãng. Khi mô hình được huấn luyện trên một tác vụ mới, nó sẽ cố gắng giữ lại những gì đã học được từ các tác vụ trước đó. Điều này được giới thiệu thông qua các kỹ thuật như hợp nhất trọng lượng đàn hồi (Elastic Weight Consolidation - EWC) và Synaptic Intelligence (SI). Các phương pháp này sẽ cố gắng giữ lại những gì đã học được từ các tác vụ trước đó bằng cách giữ lại trọng số của mô hình. Tuy nhiên, các phương pháp này không thể giữ lại được những gì đã học được từ các tác vụ trước đó nếu các tác vụ đó không có tính chất chung.

3.1.3 Các khía cạnh trong Continual Learning

Mặc dù *Catastrophic Forgetting* là một trong những khía cạnh trọng tâm trong CL. Tuy nhiên, CL còn có nhiều khía cạnh khác cần được giải quyết đến. Một số chúng có thể được đề cập đến như:

Beyond Forgetting (Vượt xa quên lãng)

Beyond Forgetting: Việc giữ lại các giá trị đã được học trước đó không chỉ là một phần quan trọng để thực hiện tốt các tác vụ trước đó. Nó cũng còn có thể sử dụng để thực hiện tốt hơn cho các tác vụ sắp đến. Điều này có nghĩa là các giá trị đã được học trước đó có thể được sử dụng để giúp mô hình học tập nhanh hơn cho các tác vụ sắp đến. Điều này còn được gọi là *Transfer Learning*.

Biological perspective (Khía cạnh sinh học)

Biological perspective: được đề cập đến việc nhìn nhận và học hỏi từ cách mà hệ thống não con người xử lý thông tin, học tập từ các kinh nghiệm và duy trì kiến thức theo thời gian. Điều này có thể được thực hiện bằng cách nghiên cứu các cơ chế sinh học của não bộ và các hệ thống học tập liên tục khác. Sự hiểu biết về cách não bộ hoạt động có thể cung cấp cơ sở để phát triển các mô hình học máy mà ở đó chúng cố gắng mô phỏng lại các cơ chế sinh học của não bộ.

3.2 Test Production

3.2.1 Giới thiệu

Production trong ngữ cảnh của Machine Learning (ML) có thể được hiểu là việc triển khai mô hình ML vào môi trường thực tế. Điều này có thể được thực hiện bằng cách sử dụng một số phương pháp như: *API, Web Application, Mobile Application, ...* Tuy nhiên, việc triển khai mô hình ML vào môi trường thực tế không phải là điều dễ dàng.

Tại sao Test Production là một sự cần thiết?

- **Độ chính xác của mô hình**: Độ chính xác của mô hình ML có thể bị giảm khi triển khai mô hình vào môi trường thực tế. Điều này có thể xảy ra do nhiều nguyên nhân khác nhau như: dữ liệu thực tế có thể không giống với dữ liệu huấn luyện, môi trường thực tế có thể không giống với môi trường huấn luyện, ...
- **Hiệu suất của mô hình**: Hiệu suất của mô hình ML có thể bị giảm khi triển khai mô hình vào môi trường thực tế. Điều này có thể xảy ra do nhiều nguyên nhân khác nhau như: mô hình có thể không đáp ứng được yêu cầu về thời gian thực, mô hình có thể không đáp ứng được yêu cầu về tài nguyên, ...
- **Tránh các cuộc tấn công bất lợi**: Các mô hình thử nghiệm có thể giúp phát hiện các cuộc tấn công bất lợi có thể xảy ra. Thay vì để cuộc tấn công này xảy ra trong môi trường sản xuất, một mô hình có thể được thử nghiệm bằng các ví dụ đối nghịch để tăng tính mạnh mẽ của nó trước khi triển khai

3.2.2 Các vấn đề trong Test Production

Môi trường thử nghiệm (Test Environment)

Đối với các nhà phát triển phần mềm truyền thống như trên các trang web các phần mềm quản lý, họ sẽ viết các mã code để tạo ra các hành vi xác định. Kiểm tra, xác định rõ các lỗi, xác định vị trĩ mã lỗi. Hầu như bao phủ tương đối các trường hợp có thể xảy ra. Tuy nhiên, với các mô hình học máy, các trường hợp có thể xảy ra là vô hạn. Vì vậy, việc kiểm tra các mô hình học máy là một vấn đề khó khăn. Để giải quyết vấn đề này, các nhà phát triển phần mềm cần phải xác định các trường hợp có thể xảy ra và tạo ra các dữ liệu tương ứng với các trường hợp đó. Tuy nhiên, việc xác định các trường hợp có thể xảy ra là một vấn đề khó khăn. Quá trình này cũng đặt ra một số thách thức như:

- Xác định kết quả đầu ra của mô hình: Trên thực tế, có rất nhiều mô hình có thể không tạo ra mô hình tương tự sau khi đào tạo lại khi dựa vào các thuật toán ngẫu nhiên.
- Thiếu đi tính minh bạch: Thường hoạt động như một hộp đen, các mô hình học máy không có tính minh bạch. Điều này có thể dẫn đến việc khó khăn trong việc xác định các trường hợp có thể xảy ra. (hộp đen là một hệ thống mà người dùng chỉ có thể xem đầu vào và đầu ra của nó mà không biết được cách nó hoạt động bên trong)
- Tính tổng quát hóa: Các mô hình học máy cần phải có tính nhất quán trong các môi trường khác ngoài môi trường huấn luyện.
- Phạm vi bao phủ: Các mô hình học máy cần phải được kiểm tra trên nhiều trường hợp có thể xảy ra.
- Nhu cầu về tài nguyên: Các mô hình học máy cần phải được kiểm tra trên nhiều trường hợp có thể xảy ra. Điều này có thể dẫn đến việc tốn nhiều tài nguyên.

3.2.3 Đánh giá mô hình

Để kiểm tra xem một mô hình có hoạt động tốt hay không, chúng ta cần phải đánh giá mô hình đó thông qua các hiệu suất của nó. Việc đánh giá mô hình là một cách tuyệt vời để đánh giá cái phiên bản khác nhau của mô hình huấn luyyeenj. Điều này có thể giúp chúng ta chọn ra phiên bản tốt nhất của mô hình. Các phương pháp kiểm thử không chỉ đánh giá hiệu suất mô hình từ các tập dữ liệu con. Nó đảm bảo rằng

các bộ phận tổng hợp của hệ thống ML đang hoạt động hiệu quả để đạt được mức kết quả chất lượng mong muốn.

Một số quy tắc kiểm thử mô hình

- Kiểm thử sau khi có một thành phần, một dữ liệu, hoặc một mô hình mới được thêm vào sau khi mô hình được huấn luyện lại (retrained).
- Kiểm thử trước khi triển khai mô hình vào môi trường thực tế.
- Viết các bài kiểm thử để kiểm tra các thành phần của mô hình.
- Viết các bài kiểm thử để kiểm tra, tránh các lỗi có thể xảy ra trong tương lai.
- Tính bền vững của mô hình yêu cầu phải tạo ra một hiệu suất tương đối ổn định ngay cả trong trường hợp có sự thay đổi căn bản về dữ liệu và mối quan hệ theo thời gian thực bằng một số cách như thêm dữ liệu mới, thay đổi dữ liệu, kiểm tra độ lệch, độ nhiễu của mô hình, ...
- Duy trì khả năng thích nghi giúp hiểu rõ hơn về các khía cạnh của mô hình (đầu vào, đầu ra, các thành phần, các thành phần phụ thuộc, các thành phần độc lập, các thành phần có thể thay đổi, các thành phần không thể thay đổi, ...)
- Khả năng tái thiết lập mô hình giúp đảm bảo rằng mô hình có thể được tái thiết lập lại một cách dễ dàng khi điều chỉnh các tham số, các thành phần,...

3.2.4 Các phương pháp kiểm thử mô hình

Một số phương pháp kiểm thử mô hình có thể được đề cập đến như:

- Invariance Testing (Kiểm thử không đổi): Kiểm tra xem mô hình có thể đưa ra các dự đoán chính xác khi dữ liệu thay đổi không?
- Direction exception testing (Kiểm thử kỳ vọng định hướng): Kiểm tra xem mô hình có thể đưa ra chính xác kỳ vọng định hướng để xác định những thay đổi về phân phối đầu vào sẽ ảnh hưởng đến đầu ra như thế nào.
- Minimum functionality testing (Kiểm thử chức năng tối thiểu): Kiểm tra chức năng tối thiểu giúp quyết định xem các thành phần mô hình riêng lẻ có hoạt động như mong đợi hay không?