|  |  |  |
| --- | --- | --- |
|  | BAN CƠ YẾU CHÍNH PHỦ  “HỌC VIỆN KỸ THUẬT MẬT MÃ” | Mẫu 2 |

BÁO CÁO CHUYÊN ĐỀ SỐ 4.6.2

“Lựa chọn tham số phù hợp cho mô hình CNN”

NHIỆM VỤ: “Nghiên cứu và ứng dụng nền tảng học sâu để xây dựng hệ thống phát hiện mã độc trực tuyến”.

Mã số: 06/2022/CB.

Cơ quan chủ trì: Học viện Kỹ thuật Mật mã

Chủ nhiệm: ThS. Lê Đức Thuận

Hà Nội - 2023

|  |  |  |
| --- | --- | --- |
|  | BAN CƠ YẾU CHÍNH PHỦ  “HỌC VIỆN KỸ THUẬT MẬT MÃ” |  |

BÁO CÁO CHUYÊN ĐỀ SỐ 4.6.2

“Lựa chọn tham số phù hợp cho mô hình CNN”

NHIỆM VỤ: “Nghiên cứu và ứng dụng nền tảng học sâu để xây dựng hệ thống phát hiện mã độc trực tuyến”.

Mã số: 06/2022/CB.

Cơ quan chủ trì: Học viện Kỹ thuật Mật mã

Chủ nhiệm: ThS. Lê Đức Thuận

|  |  |
| --- | --- |
| **Người thực hiện chuyên đề** | **Cơ quan chủ trì** |
| *(Họ tên và chữ ký)* | *(Họ tên và chữ ký)* |

Hà Nội - 2023

MỤC LỤC

[MỤC LỤC 1](#_Toc129699069)

[DANH MỤC HÌNH ẢNH 1](#_Toc129699070)

[LỰA CHỌN CÁC THAM SỐ PHÙ HỢP TRONG MÔ HÌNH CNN 2](#_Toc129699071)

[1.1. Các tham số trong mô hình mạng CNN 2](#_Toc129699072)

[1.1.1. Lớp tích chập 2](#_Toc129699073)

[1.1.2. Lớp tổng hợp 3](#_Toc129699074)

[1.1.3. Lớp kết nối đầy đủ 4](#_Toc129699075)

[1.1.4. Hàm mất mát (loss) 4](#_Toc129699076)

[1.1.5. Tối ưu hóa 5](#_Toc129699077)

[1.2. Code thiết lập tham số trong CNN trong bài toán phát hiện mã độc 6](#_Toc129699078)

[TÀI LIỆU THAM KHẢO 11](#_Toc129699079)

# DANH MỤC HÌNH ẢNH

[Hình 1: Sơ đồ hoạt động của hàm kích hoạt 3](#_Toc129699058)

[Hình 2: Cơ chế hoạt động tổng hợp trung bình, tổng hợp cực đại 4](#_Toc129699059)

[Hình 3: So sánh Adam và các thuật toán tối ưu khác trong việc huấn luyện đa lớp 6](#_Toc129699060)

# LỰA CHỌN CÁC THAM SỐ PHÙ HỢP TRONG MÔ HÌNH CNN

## Các tham số trong mô hình mạng CNN

Mạng nơ-ron tích chập là một loại mạng học sâu phổ biến có các đại diện nổi tiếng như AlexNet, VGG, GoogleNet [1-4]. Mạng nơ-ron tích chập được nghiên cứu và ứng dụng thành công trong các bài toán nhận dạnghình ảnh và giọng nói nhờ khả năng trích xuất đặc trưng cục bộ và toàn thể thông qua nhân tích chập [5]. Cấu trúc mạng nơ-ron tích chập gồm 2 thành phần là tổng hợp đặc trưng và phân lớp. Thành phần tổng hợp đặc trưng bao gồm một hoặc nhiều cặp lớp tích chập và lớp tổng hợp.

### Lớp tích chập

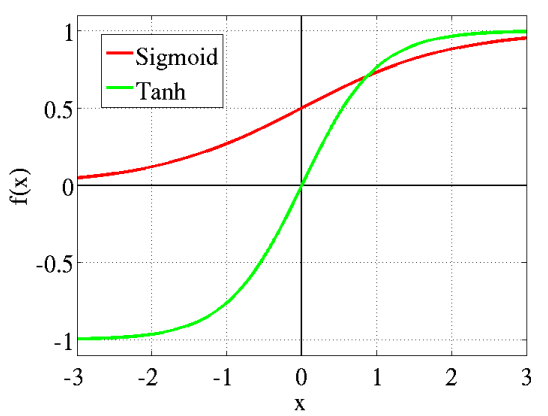
Đối với mỗi lớp tích chập: Các lớp được liên kết với nhau thông qua cơ chế tích chập, lớp tiếp theo là kết quả của phép tích chập ở lớp trước đó, nhờ vậy mà ta có các kết nối cục bộ.

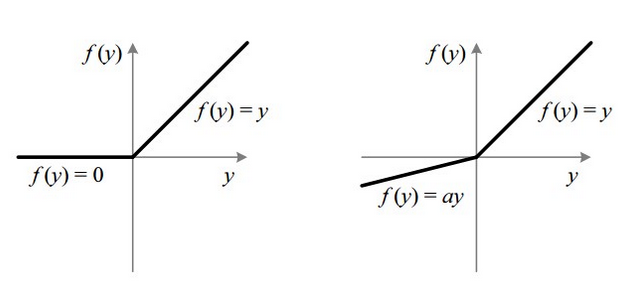
Mỗi lớp tích chập sử dụng hàm kích hoạt trước khi đưa ra kết quả. Nếu như không có hàm kích hoạt toàn bộ mạng nơ-ron sẽ chuyển dịch tuyến tính từ đầu vào cho tới đầu ra. Đối với mạng tích chập thì thường sử dụng một số hàm kích hoạt như: Sigmoid, TanH (Hyperbolic Tangent), ReLU (Rectified Linear Unit), leaky ReLU,… Được thể hiện như Hình 1.

Khi kết thúc lớp tích chập sử dụng hàm kích hoạt ReLU, ma trận sẽ được cập nhật theo công thức (1).

|  |  |
| --- | --- |
|  | (1) |

Relu giúp cho giá trị của hàm số tiến tới 0 và +∞ nhanh hơn, quá trình huấn luyện sẽ nhanh hơn.





Hình 1: Sơ đồ hoạt động của hàm kích hoạt

### Lớp tổng hợp

Các đặc trưng được hiệu chỉnh khi đi qua lớp tích chập sẽ tiếp tục đi vào lớp tổng hợp (pooling) để chắt lọc các thông tin hữu ích với bài toán và giảm bớt số lượng nơron. Công việc được thực hiện trong lớp này là tìm ra giá trị đại diện cho một vùng ma trận của lớp trước, qua đó các dữ liệu thừa sẽ được loại bỏ tránh trường hợp học lệch.

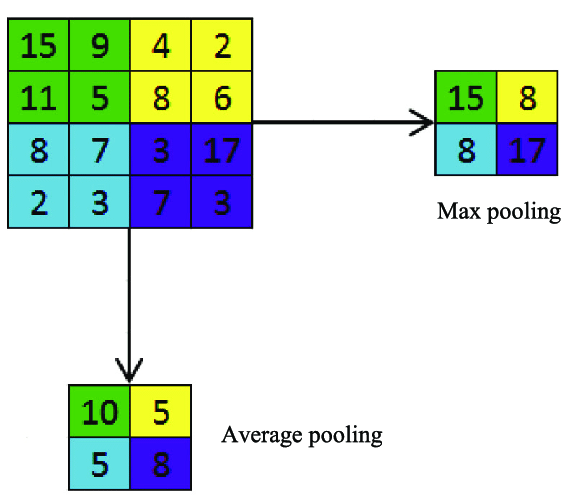
Có hai kiểu tổng hợp được sử dụng rộng rãi là tổng hợp trung bình và tổng hợp cực đại. Tổng hợp cực đại lựa chọn phần tử lớn nhất trong mỗi nhóm trong khi đó tổng hợp trung bình lấy trung bình tất cả các giá trị trong nhóm như minh họa trong Hình 2. Theo công thức sau [15]:

xkpq (2)

 xkpq (3)

Với

* : Giá trị đầu ra thứ k của liên kết ánh xạ đặc trưng
* : ký hiệu phần tử tại vị trí *(p, q)* được chứa trong vùng pooling .



Hình 2: Cơ chế hoạt động tổng hợp trung bình, tổng hợp cực đại

Do đó, nếu ma trận tổng hợp là 2x2 thì kích thước của lớp tích chập sẽ giảm đi 4 lần.

### Lớp kết nối đầy đủ

Lấy toàn bộ các nơ-ron đầu ra của lớp trước đó (lớp tích chập và lớp tổng hợp) làm nơ-ron đầu vào, hay còn gọi là lớp làm phẳng (flatten). Mỗi nơ-ron của lớp làm phẳng được kết nối đầy đủ với tất cả nơ-ron của lớp tiếp theo. Bản chất của lớp kết nối đầy đủ chính là một mạng nơ-ron cơ bản gồn n lớp ẩn. Lớp ngoài cùng (output) là quá trình phân lớp, quyết định xem tập tin này nó có nhãn nào. Quyết định phân lớp này thường sử dụng hàm kích hoạt *Softmax*. Công thức hàm *softmax* tính cho đầu ra thứ *j* như sau:

 (4)

Hàm *softmax* tính xác xuất của từng từng đầu ra trong lớp đầu ra dựa trên các nơ-ron và trọng số tương ứng của lớp kết nối đầy đủ trước đó. Giá trị của từng đầu ra nằm trong [0,1], tổng của các giá trị đầu ra bằng 1. Lấy giá trị lớn nhất của *softmax* để quyết định đó là nhãn của đầu ra.

### Hàm mất mát (loss)

Hàm mất mát được sử dụng để đánh giá hiệu suất hoạt động của mô hình. Giá trị của hàm mất mát càng thấp thì chất lượng của mô hình càng cao và ngược lại.

Có nhiều hàm mất mát, nhưng trong đó Entropy chéo thường được sử dụng nhất trong các mô hình học sâu với bài toán phân lớp. Khi đánh giá sự khác nhau (hay khoảng cách) giữa hai phân bố xác suất chúng ta sử dụng Entropy chéo sẽ cho kết quả hiệu quả hơn.

Entropy chéo giữa hai phân phối*p* và *q***:**

 (5)

Đối với Softmax Regression, hàm mất mát được định nghĩa như sau:

 Với *yji* và aji lần lượt là phần tử thứ i của vector (xác suất) *yi*và *ai*. **W** là ma trận trọng số cần đươc tối ưu.

Categorical crossentropy và Sparse categorical crossentropy là hai hàm mất mát thường được sử dụng khi biên dịch một mô hình cho bài toán phân lớp. Hàm mất mát được sử dụng trong suốt quá trình huấn luyện. Nó sẽ dựa vào sự mất mất so với thực tế để cải thiện, cập nhật lại trọng số nhằm tăng độ chính xác để gần hơn so với nhãn thực tế.

### Tối ưu hóa

Thuật toán Gradient Descent (GD) là một trong nhiều phương pháp được sử dụng để cải thiện trọng số trong quá trình huấn luyện. Công thức thuật toán GD có thể được viết như sau:

 (7)

Chúng ta cần di chuyển *θ* sao cho *J(θ)* đạt giá trị nhỏ nhất. ***α*** là tốc độ học (learning rate). Mục đích chính của việc tối ưu hóa là tính toán tốc độ học cho phù hợp với mỗi tham số mô hình.

Các hàm tối ưu thường được sử dụng trong qua trình huấn luyện mô hình:

* Momentum
* Adagrad
* RMSprop
* Adam
* AdaMax
* Nadam

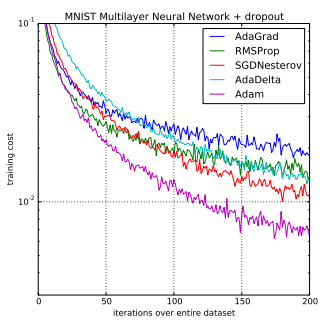
Trong bài viết này, chúng tôi sẽ tập trung vào phương pháp tối ưu **Adam.**

Ta có công thức của phương pháp Adam như sau:

 (8)

Hai công thức trên thể hiện hai đặc tính mà Adam kế thừa từ 2 phương pháp tối ưu khác là Adadelta và RMSprop:

* Duy trì tốc độ học trên mỗi tham số, giúp cải thiện hiệu suất với các vấn đề liên quan đến sparse gradient
* Duy trì tốc độ học, được điều chỉnh dựa trên mức trung bình của gradient đối với trọng số.



Hình 3: So sánh Adam và các thuật toán tối ưu khác trong việc huấn luyện đa lớp

## Code thiết lập tham số trong CNN trong bài toán phát hiện mã độc

import numpy as np

import pandas as pd

import csv

import keras

from keras.datasets import mnist

from keras.models import Sequential

from keras.layers import Dense, Dropout, Flatten

from keras.layers import Conv2D, MaxPooling2D

from keras import backend as K

train\_df = pd.read\_csv(r'D:\Phát hiện mã độc bằng học sâu\phân tích code CNN\test\10setPermissionApi\train-9.csv') #đưa vao tap du lieu chung, ko can chia train va test

#train\_df=train\_df.sample(100)

train\_data1 = np.array(train\_df, dtype='u1')

test\_df = pd.read\_csv(r'D:\Phát hiện mã độc bằng học sâu\phân tích code CNN\test\10setPermissionApi\file-9.csv') #đưa vao tap du lieu chung, ko can chia train va test

#test\_df=test\_df.sample(100)

test\_data1 = np.array(test\_df, dtype='u1')

val\_df = pd.read\_csv(r'D:\Phát hiện mã độc bằng học sâu\phân tích code CNN\test\10setPermissionApi\file-0.csv') #đưa vao tap du lieu chung, ko can chia train va test

#val\_df=val\_df.sample(100)

val\_data1 = np.array(val\_df, dtype='u1')

# Đưa dữ liệu vào mảng

train\_y = train\_data1[1:, 0]

train\_data= train\_data1[1:,1:-5]

print(train\_data.shape)

'''

val\_y = val\_data1[1:, 0]

val\_data= val\_data1[1:,1:-5]

test\_y = test\_data1[1:, 0]

test\_data= test\_data1[1:,1:-5]

train\_x1 = train\_data[:,:398]

train\_x2 = train\_data[:,398:598]

print("so chieu cua train1: ", train\_x1.shape)

print("so chieu cua train2: ", train\_x2.shape)

val\_x1 = val\_data[:,:398]

val\_x2 = val\_data[:,398:]

print("so chieu cua val1: ", val\_x1.shape)

print("so chieu cua val2: ", val\_x2.shape)

test\_x1 = test\_data[:,:398]

test\_x2 = test\_data[:,398:598]

print("so chieu cua test1: ", test\_x1.shape)

print("so chieu cua test2: ", test\_x2.shape)

train\_x1=np.pad(train\_x1,((0,0),(0,2)),"constant")

val\_x1= np.pad(val\_x1,((0,0),(0,2)),"constant")

test\_x1=np.pad(test\_x1,((0,0),(0,2)),"constant")

train\_x2=np.pad(train\_x2,((0,0),(0,25)),"constant")

val\_x2= np.pad(val\_x2,((0,0),(0,25)),"constant")

test\_x2=np.pad(test\_x2,((0,0),(0,25)),"constant")

'''

train\_x1 = train\_data[:,:398]

train\_x2 = train\_data[:,398:598]

print("so chieu cua train1: ", train\_x1.shape)

print("so chieu cua train2: ", train\_x2.shape)

concat\_test\_val = np.concatenate((test\_data1,val\_data1))

test\_y = concat\_test\_val[1:, 0]

test\_data= concat\_test\_val[1:,1:-5]

test\_x1 = test\_data[:,:398]

test\_x2 = test\_data[:,398:598]

print("so chieu cua test1: ", test\_x1.shape)

print("so chieu cua test2: ", test\_x2.shape)

train\_data=np.pad(train\_data,((0,0),(0,27)),"constant")

test\_data=np.pad(test\_data,((0,0),(0,27)),"constant")

train\_x1=np.pad(train\_x1,((0,0),(0,2)),"constant")

test\_x1=np.pad(test\_x1,((0,0),(0,2)),"constant")

print("so chieu cua train\_x1: ", train\_x1.shape)

train\_x2=np.pad(train\_x2,((0,0),(0,25)),"constant")

test\_x2=np.pad(test\_x2,((0,0),(0,25)),"constant")

# Khởi tạo hằng số

BATCH\_SIZE = 64 #số lượng các ví dụ huấn luyện trong 1 đợt. Kích thước càng lớn thì càng cần bộ nhớ nhiều.

IMG\_SIZE1 = 25

IMG\_SIZE2 = 15

N\_CLASSES = 180

LR = 0.001

N\_EPOCHS = 100

train\_data = train\_data.reshape(-1, IMG\_SIZE1, IMG\_SIZE1, 1)

#val\_x1 = val\_x1.reshape(-1, IMG\_SIZE1, IMG\_SIZE1, 1)

test\_data = test\_data.reshape(-1, IMG\_SIZE1, IMG\_SIZE1, 1)

'''

#ĐƯa dữ liệu về dạng phù hợp

train\_x1 = train\_x1.reshape(-1, IMG\_SIZE1, IMG\_SIZE1, 1)

#val\_x1 = val\_x1.reshape(-1, IMG\_SIZE1, IMG\_SIZE1, 1)

test\_x1 = test\_x1.reshape(-1, IMG\_SIZE1, IMG\_SIZE1, 1)

train\_x2 = train\_x2.reshape(-1, IMG\_SIZE2, IMG\_SIZE2, 1)

#val\_x2 = val\_x2.reshape(-1, IMG\_SIZE2, IMG\_SIZE2, 1)

test\_x2 = test\_x2.reshape(-1, IMG\_SIZE2, IMG\_SIZE2, 1)

'''

# convert class vectors to binary class matrices

train\_y = keras.utils.to\_categorical(train\_y, N\_CLASSES)

test\_y = keras.utils.to\_categorical(test\_y, N\_CLASSES)

#val\_y = keras.utils.to\_categorical(val\_y, N\_CLASSES)

model = Sequential()

model.add(Conv2D(32, kernel\_size=(3, 3),

activation='relu', padding='same',

input\_shape=(IMG\_SIZE1, IMG\_SIZE1, 1)))

model.add(MaxPooling2D(pool\_size=(2, 2), padding='same'))

model.add(Conv2D(32, (3, 3), activation='relu', padding='same'))

model.add(MaxPooling2D(pool\_size=(2, 2), padding='same'))

model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))

model.add(MaxPooling2D(pool\_size=(2, 2), padding='same'))

model.add(Dropout(0.25))

model.add(Flatten())

model.add(Dense(1024, activation='relu'))

model.add(Dense(N\_CLASSES, activation='softmax'))

model.summary()

model.compile(loss=keras.losses.categorical\_crossentropy,

optimizer=keras.optimizers.Adadelta(),

metrics=['accuracy'])

model.fit(train\_data, train\_y,

batch\_size=BATCH\_SIZE,

epochs=N\_EPOCHS,

verbose=1,

validation\_data=(test\_data, test\_y))

score = model.evaluate(test\_data, test\_y, verbose=0)

print('Test loss:', score[0])

print('Test accuracy:', score[1])

# TÀI LIỆU THAM KHẢO

1. A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in Advances in Neural Information Processing Systems 25 (NIPS 2012), Harrahs and Harveys, Lake Tahoe, 2012, pp. 1097-1105.
2. A. Z. K. Simonyan, "Very Deep Convolutional Networks for LargeScale Image Recognition," in International Conference on Learning Representations 2015 (ICLR2015), San Diego, CA, 2015.
3. C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna.,”Rethinking the Inception Architecture for Computer Vision.”in Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), WA, USA, June 2016.
4. K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, Nevada, 2016.
5. Ganesh, Meenu & Pednekar, Priyanka & Prabhuswamy, Pooja & Sreedharan Nair, Divyashri & Park, Younghee & Jeon, Hyeran. (2017). CNN-Based Android Malware Detection. 60-65. 10.1109/ICSSA.2017.18.
6. <https://www.sec.cs.tu-bs.de/~danarp/drebin/index.html>
7. Daniel Arp, Michael Spreitzenbarth, Malte Huebner, Hugo Gascon, and Konrad Rieck ["Drebin: Efficient and Explainable Detection of Android Malware in Your Pocket"](https://www.tu-braunschweig.de/Medien-DB/sec/pubs/2014-ndss.pdf), 21th Annual Network and Distributed System Security Symposium (NDSS), February 2014
8. Dong-Jie Wu, ChingHao Mao, Te-En Wei, HahnMing Lee and Kuo-Ping Wu, “DroidMat: Android Malware Detection through Manifest and API Calls Tracing”, in [2012 Seventh Asia Joint Conference on Information Security](https://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=6296578), Tokyo Japan, 2012.
9. Wenbo Yang, Yuanyuan Zhang, Junliang Shu, Bodong Li, Wenjun Hu, Dawu Gu, “AppSpear: Bytecode Decrypting and DEX Reassembling for Packed Android Malware”, in Bos H., Monrose F., Blanc G. (eds) Research in Attacks, Intrusions, and Defenses. RAID 2015. Lecture Notes in Computer Science, vol 9404. Springer, Cham
10. Asaf Shabtai, Uri Kanonov, Yuval Elovici, Chanan Glezer, Yael Weiss, “Androimaly: a behavioral malware detection framework for android devices”, in Journal of Intelligent Information Systems, Volume 38, Isssue 1, February 2012, ISSN 1573-7675, pp 161-190.
11. Smita Ranveer and Swapnaja Hiray, "Comparative Analysis of Feature Extraction Methods of Malware Detection", International Journal of Computer Applications, Volume 120 - No. 5, June 2015.
12. Aziz Makandar and Anita Patrot, "Malware analysis and classification using Artificial Neural Network", International Conference on Trends in Automation, Communications and Computing Technology, 2015, pp 1-6.
13. Komal Narang, "Accuracy comparison among different machine learning techniques for detecting malicious codes", Proceedings of AIP Conference, 2015.
14. Lê Đức Thuận, Phạm Ngọc Hưng, Phạm Văn Hưởng, Nguyễn Đức Trung, “Phát hiện mã độc Android dựa trên mạng niềm tin sâu”, FAIR 2018.
15. 2. Waseem Rawat and Zenghui Wang, Deep Convolutional Neural Networks for Image Classification: A Comprehensive Review, Neural Computation, volume 29, Issue 0, September 2017, p. 2352-2449