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

BÁO CÁO CHUYÊN ĐỀ SỐ 1.1

“Tìm hiểu về Keras/ Tensorflow”

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 - 2022

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

BÁO CÁO CHUYÊN ĐỀ SỐ 1.1

“Tìm hiểu về Keras/ Tensorflow”

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 - 2022

MỤC LỤC

[MỤC LỤC 1](#_Toc115448063)

[DANH MỤC HÌNH ẢNH 2](#_Toc115448064)

[TÌM HIỂU VỀ KERAS/ TENSORFLOW 3](#_Toc115448065)

[1.1. Tổng quan Keras 3](#_Toc115448066)

[1.2 Tính năng trong Keras 3](#_Toc115448067)

[1.3 Kiến trúc của Keras 4](#_Toc115448068)

[1.3.1 Model 5](#_Toc115448069)

[1.3.2 Layer 6](#_Toc115448070)

[1.3.3 Core Modules 7](#_Toc115448071)

[1.4 Các mô-đun thường dùng Keras 8](#_Toc115448072)

[1.5 Backend mô-đun trong keras 9](#_Toc115448073)

[1.6 Utils module 12](#_Toc115448074)

# DANH MỤC HÌNH ẢNH

[Hình 1.1:Sơ đồ mô tả quan hệ giữa mô hình, lớp và core mô đun 5](#_Toc115448172)

# TÌM HIỂU VỀ KERAS/ TENSORFLOW

Tensorflowlà một thư viện dạng nguồn mở được tạo ra và phát triển bởi đội ngũ chuyên viên của Google. Nó được sử dụng rất phổ biến trong lĩnh vực học máy - Machine Learning và học sâu - Deep Learning hỗ trợ mạnh mẽ các phép toán học để tính toán.

Hiện nay, Tensorflow cơ bản được xem như một trong những phương tiện trung gian giúp tính toán cho các số lượng có trong sản xuất. Nó phục vụ cho nhu cầu học tập cũng như nghiên cứu một cách dễ dàng hơn và đồng thời trở thành một công cụ không thể thiếu trong Machine Learning.

## 1.1. Tổng quan Keras

Keras là một API cấp cao được thiết kế cho Python để triển khai mạng nơ-ron dễ dàng hơn. Có thể hiểu đơn giản Keras là một thư viện “high level” với phần “low-level” là TensorFlow.

Khi được nhúng vào TensorFlow, nó cung cấp các mô-đun có sẵn cho tất cả các tính toán mạng nơ-ron và do đó có thể thực hiện học sâu rất nhanh. TensorFlow rất linh hoạt và lợi ích chính là tính toán phân tán. Sử dụng Keras, bạn có thể linh hoạt và có thể kiểm soát ứng dụng của mình, thực hiện ý tưởng của bạn trong thời gian ngắn. Trong khi tính toán liên quan đến tensors, đồ thị tính toán, phiên, v.v. có thể được tùy chỉnh bằng cách sử dụng Tensorflow Core API.

## 1.2 Tính năng trong Keras

Keras rất mạnh mẽ và năng động, ta có thể sử dụng chung với các thư viện nổi tiếng như Tensorflow, CNTK, Theano. Nó tận dụng các kỹ thuật tối ưu hóa khác nhau để làm cho API mạng thần kinh cấp cao dễ dàng hơn và hiệu quả hơn. Nó hỗ trợ các tính năng sau

* API nhất quán, đơn giản và có thể mở rộng.
* Cấu trúc tối thiểu - dễ dàng đạt được kết quả mà không cần rườm rà.
* Hỗ trợ nhiều nền tảng và backend.
* Thân thiện với người dùng chạy trên cả CPU và GPU.
* Khả năng mở rộng tính toán cao.

Keras năng động , mạnh mẽ và có những ưu điểm sau

* Hỗ trợ cộng đồng rộng lớn với nhiều người dùng và tài liệu, sẵn sàng trợ giúp hơn các khuôn khổ học sâu khác.
* Dễ dàng để kiểm tra.
* Mạng nơ-ron Keras được viết bằng Python giúp mọi thứ đơn giản hơn.
* Keras hỗ trợ cả mạng convolution và recurrent.
* Mô hình học sâu là các thành phần rời rạc, do đó, bạn có thể kết hợp thành nhiều cách.
* Đối với những người mới tiếp cận đến Deep Learning thì chọn sử dụng Keras để build model vì nó đơn giản,dễ nắm bắt hơn các thư viện khác.

Đây là một công cụ đơn giản nhưng linh hoạt dùng để nghiên cứu sáng tạo. Các developers sử dụng nó để tạo các mô hình sâu cho điện thoại thông minh, để đào tạo phân tán các mô hình học sâu. Ngoài ra, họ có thể sử dụng nó để tạo và triển khai mô hình làm việc trong thời gian ngắn.

Các mô hình Keras của bạn có thể dễ dàng được triển khai trên nhiều nền tảng hơn bất kỳ API học sâu nào khác:

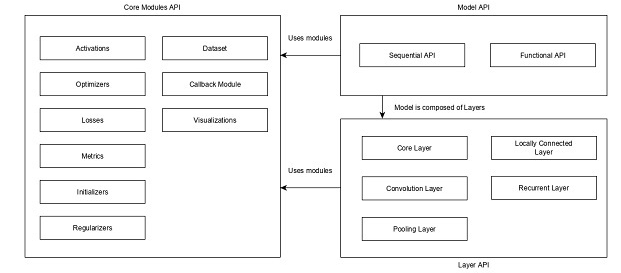
* Trên máy chủ thông qua thời gian chạy Python hoặc thời gian chạy Node.js
* Trên máy chủ thông qua TFX / TF Serving
* Trong trình duyệt qua TF.js
* Trên Android hoặc iOS thông qua TF Lite hoặc CoreML của Apple
* Trên Raspberry Pi, trên Edge TPU hoặc một hệ thống nhúng khác

## 1.3 Kiến trúc của Keras

API Keras có thể được chia thành ba loại chính:

* Model
* Layer
* Core Modules

Trong Keras, mọi ANN đều được đại diện bởi Keras Models. Đổi lại, mọi Mô hình Keras là thành phần của các Lớp Keras và đại diện cho các lớp ANN như input, hidden layer, output layers, convolution layer, pooling layer, v.v., mô hình Keras và các mô-đun truy cập lớp Keras cho activation function, loss function, regularization function ,v.v., Sử dụng mô hình Keras, Lớp Keras và mô-đun Keras, bất kỳ thuật toán ANN nào (CNN, RNN, v.v.,) đều có thể được biểu diễn theo cách đơn giản và hiệu quả.



Hình 1.1:Sơ đồ mô tả quan hệ giữa mô hình, lớp và core mô đun

### 1.3.1 Model

**API mô hình tuần tự (sequential)**

Sequential (Mô hình tuần tự**)** về cơ bản là một thành phần tuyến tính của các Lớp Keras. Mô hình tuần tự rất dễ dàng, tối thiểu cũng như có khả năng biểu diễn gần như tất cả các mạng nơ-ron có sẵn.

Một mô hình tuần tự đơn giản như sau:

**from** keras.models **import** Sequential

**from** keras.layers **import** Dense, Activation

model = Sequential()

model.add(Dense(512, activation = 'relu', input\_shape = (784,)))

* Dòng 1 nhập mô hình tuần tự từ mô hình Keras
* Dòng 2 nhập Dense layer và Activation module
* Dòng 4 tạo một mô hình tuần tự mới bằng cách sử dụng Sequential API
* Dòng 5 thêm một dense layer (Dense API) với relu activation (sử dụng mô-đun Kích hoạt).

Mô hình tuần tự cho thấy lớp Model để tạo ra các mô hình tùy chỉnh.Ta có thể sử dụng khái niệm sub-classing để tạo mô hình phức tạp của riêng mình.

**API chức năng (Functional)**

API chức năng là một phương pháp để tạo ra các mô hình linh hoạt hơn API mô hình tuần tự. Nó có thể xử lý các mô hình có cấu trúc liên kết phi tuyến tính, các lớp chia sẻ và thậm chí là nhiều đầu vào hoặc đầu ra.

Ý tưởng chính là mô hình học sâu thường là một đồ thị xoay chiều có hướng (DAG) của các lớp. Vì vậy, API chức năng là một cách để xây dựng đồ thị của các lớp .

import numpy as np

import tensorflow as tf

from tensorflow import keras

from tensorflow.keras import layers

### 1.3.2 Layer

Mỗi lớp Keras trong mô hình Keras đại diện cho lớp tương ứng (input layer, hidden layer and output layer) trong mô hình mạng nơ ron được đề xuất thực tế. Keras cung cấp rất nhiều lớp tiền xây dựng để có thể dễ dàng tạo ra bất kỳ mạng nơ-ron phức tạp nào. Một số lớp Keras quan trọng được chỉ định bên dưới,

* Core Layers
* Convolution Layers
* Pooling Layers
* Recurrent Layers

Mô hình mạng thần kinh sử dụng mô hình tuần tự như sau :

**from** keras.models import Sequential

**from** keras.layers import Dense, Activation, Dropout model = Sequential()

model.**add**(Dense(512, activation = 'relu', input\_shape = (784,)))

model.**add**(Dropout(0.2))

model.**add**(Dense(512, activation = 'relu')) model.**add**(Dropout(0.2))

model.**add**(Dense(num\_classes, activation = 'softmax'))

* Dòng 1 nhập Sequential model từ mô hình Keras
* Dòng 2 nhập Dense layer và Activation module
* Dòng 4 tạo một mô hình tuần tự mới bằng cách sử dụng Sequential API
* Dòng 5 thêm một dense layer (Dense API) vớ hàmi relu activation (sử dụng Activation module).
* Dòng 6 thêm một lớp dropout layer (Dropout API) để xử lý over-fitting.
* Dòng 7 thêm một lớp dense layer (Dense API) với hàm relu activation (sử dụng Activation module).
* Dòng 8 thêm một lớp dropout layer (Dropout API) để xử lý over-fitting.
* Dòng 9 thêm lớp dense layer (Dense API) với hàm softmax activation (sử dụng Activation module).

Keras cũng cung cấp các tùy chọn để tạo các lớp có thể tùy chỉnh. Lớp tùy chỉnh có thể được tạo bằng cách sub-classing lớp Keras.Layer và nó tương tự như các mô hình Keras sub-classing

### 1.3.3 Core Modules

Keras cũng cung cấp rất nhiều chức năng liên quan đến mạng nơ-ron tích hợp sẵn để tạo đúng mô hình Keras và các lớp Keras. Một số chức năng như sau:

* **Activations module** − là một khái niệm quan trọng trong ANN và mô-đun kích hoạt cung cấp nhiều chức năng kích hoạt như softmax, relu, v.v.,
* **Loss module** − cung cấp các hàm loss như mean\_squared\_error, mean\_absolute\_error, poisson, v.v.,
* **Optimizer module** − cung cấp chức năng trình tối ưu hóa như adam, sgd, v.v.,
* **Regularizers** − cung cấp các chức năng như bộ điều chỉnh L1, bộ điều chỉnh L2, v.v.,

## 1.4 Các mô-đun thường dùng Keras

Đầu tiên ta hãy xem danh sách các mô-đun có sẵn trong Keras.

* **Initializers** − Cung cấp danh sách các chức năng khởi tạo. Ta có thể tìm hiểu chi tiết trong bài lớp Keras. trong giai đoạn tạo mô hình của học máy.
* **Regularizers** − Cung cấp danh sách chức năng bộ điều chỉnh.
* **Constraints** − Cung cấp danh sách hàm regularizers
* **Activations** − Cung cấp danh sách hàm activator
* **Losses** − Cung cấp danh sách hàm loss.
* **Metrics** − Cung cấp một danh sách các hàm số liệu
* **Optimizers** − Cung cấp danh sách hàm của trình tối ưu hóa
* **Callback** −Cung cấp danh sách hàm callback. Ta có thể sử dụng nó trong quá trình đào tạo để in dữ liệu trung gian cũng như tự dừng đào tạo (phương pháp EarlyStopping) dựa trên một số điều kiện
* **Text processing** − Cung cấp các chức năng chuyển đổi văn bản thành mảng NumPy phù hợp với học máy. Ta có thể sử dụng nó trong giai đoạn chuẩn bị dữ liệu của máy học
* **Image processing** −Cung cấp các chức năng chuyển đổi ảnh thành mảng NumPy phù hợp với máy học. Ta có thể sử dụng nó trong giai đoạn chuẩn bị dữ liệu của máy học.
* **Sequence processing** − Cung cấp các chức năng tạo dữ liệu dựa trên thời gian từ dữ liệu đầu vào đã cho. Ta có thể sử dụng nó trong giai đoạn chuẩn bị dữ liệu của máy học.
* **Backend** − Cung cấp chức năng của thư viện backend như *TensorFlow* và *Theano*.
* **Utilities** −Cung cấp nhiều chức năng tiện ích hữu ích trong học sâu

## 1.5 Backend mô-đun trong keras

Mô-đun backend được sử dụng cho các hoạt động backend của keras. Theo mặc định, keras chạy trên phần back end TensorFlow. Nếu muốn, bạn có thể chuyển sang các backend khác như Theano hoặc CNTK. Cấu hình backend Defualt được xác định bên trong thư mục gốc dưới tệp .keras / keras.json.

Mô-đun backend Keras như sau :

>>> **from** keras **import** backend **as** k

Nếu ta đang sử dụng TensorFlow backend mặc định, thì hàm dưới đây trả về thông tin dựa trên TensorFlow như sau:

>>> k.backend()

'tensorflow'

>>> k.epsilon()

1e-07

>>> k.image\_data\_format()

'channels\_last'

>>> k.floatx()

'float32'

Ngắn gọn một số hàm backend quan trọng được sử dụng để phân tích dữ liệu -

**\*get\_uid()**

Định danh cho đồ thị mặc định

>>> k.get\_uid(prefix='')

1

>>> k.get\_uid(prefix='') 2

**\*reset\_uids**

Sử dụng để đặt lại giá trị uid

>>> k.reset\_uids()

Sau đó, hãy thực thi get\_uid(). Và giá trị sẽ đặt lại là 1

>>> k.get\_uid(prefix='')

1

**\*placeholder**

Khởi tạo placebolder tensor. Placeholder đơn giản dùng để giữ hình dạng 3D như sau :

>>> data = k.placeholder(shape = (1,3,3))

>>> data

<tf.Tensor 'Placeholder\_9:0' shape = (1, 3, 3) dtype = float32>

If you use int\_shape(), it will show the shape.

>>> k.int\_shape(data) (1, 3, 3)

**\*dot**

Được sử dụng để nhân 2 tensor. Coi a và b là hai tensor và c sẽ là kết quả của phép nhân của ab. Giả sử shape a là (4,2) và shape b là (2,3):

>>> a = k.placeholder(shape = (4,2))

>>> b = k.placeholder(shape = (2,3))

>>> c = k.dot(a,b)

>>> c

<tf.Tensor 'MatMul\_3:0' shape = (4, 3) dtype = float32>

>>>

**\*ones**

Khởi tạo với tất cả các giá trị 1 :

>>> res = k.ones(shape = (2,2))

#print the value

>>> k.eval(res)

array([[1., 1.], [1., 1.]], dtype = float32)

**\*batch\_dot**

Được sử dụng để tính tích của hai dữ liệu theo bactches. Kích thước đầu vào phải từ 2 trở lên

>>> a\_batch = k.ones(shape = (2,3))

>>> b\_batch = k.ones(shape = (3,2))

>>> c\_batch = k.batch\_dot(a\_batch,b\_batch)

>>> c\_batch

<tf.Tensor 'ExpandDims:0' shape = (2, 1) dtype = float32>

**\*variable**

Sử dụng để khởi tạo biến. Ta sẽ thực hiện thao tác chuyển đổi đơn giản như sau :

>>> data = k.variable([[10,20,30,40],[50,60,70,80]])

#variable initialized here

>>> result = k.transpose(data)

>>> print(result)

Tensor("transpose\_6:0", shape = (4, 2), dtype = float32)

>>> print(k.eval(result))

[[10. 50.]

[20. 60.]

[30. 70.]

[40. 80.]]

Truy cập từ numpy bằng cách :

>>> data = np.array([[10,20,30,40],[50,60,70,80]])

>>> print(np.transpose(data))

[[10 50]

[20 60]

[30 70]

[40 80]]

>>> res = k.variable(value = data)

>>> print(res)

<tf.Variable 'Variable\_7:0' shape = (2, 4) dtype = float32\_ref>

**\*is\_sparse(tensor)**

Được sử dụng để kiểm tra xem liệu tensor có đồng đều hay không

>>> a = k.placeholder((2, 2), sparse=True)

>>> print(a) SparseTensor(indices =

Tensor("Placeholder\_8:0",

shape = (?, 2), dtype = int64),

values = Tensor("Placeholder\_7:0", shape = (?,),

dtype = float32), dense\_shape = Tensor("Const:0", shape = (2,), dtype = int64))

>>> print(k.is\_sparse(a)) True

**\*to\_dense()**

Được sử dụng để chuyển đổi tensor từ sparse thành dense

>>> b = k.to\_dense(a)

>>> print(b) Tensor("SparseToDense:0", shape = (2, 2), dtype = float32)

>>> print(k.is\_sparse(b)) False

**\*random\_uniform\_variable**

Được sử dụng để khởi bằng cách sử dụng định nghĩa phân phối đồng nhất (***uniform distribution***)

**k**.random\_uniform\_variable(shape, mean, scale)

Ở đây :

* **shape** − biểu thị các hàng và cột ở định dạng tuples
* **mean** − trung bình của phân phối đồng nhất.
* **scale** − độ lệch chuẩn của phân phối đồng nhất.

**Ví dụ :**

>>> a = k.random\_uniform\_variable(shape = (2, 3), low=0, high = 1)

>>> b = k. random\_uniform\_variable(shape = (3,2), low = 0, high = 1)

>>> c = k.dot(a, b)

>>> k.int\_shape(c)

(2, 2)

## 1.6 Utils module

utils cung cấp chức năng tiện ích hữu ích cho học sâu. Một số phương thức được cung **cấp bởi mô-đun utils như sau:**

* **HDF5Matrix**

Được sử dụng để biểu diễn dữ liệu đầu vào ở dạng HDF5

**from** keras.utils **import** HDF5Matrix data = HDF5Matrix('data.hdf5', 'data')

* **to\_categorical**

Sử dụng để chuyển đổi lớp vector từ lớp ma trận nhị phân

>>> from keras.utils import to\_categorical

>>> labels = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

>>> to\_categorical(labels)

array([[1., 0., 0., 0., 0., 0., 0., 0., 0., 0.],

[0., 1., 0., 0., 0., 0., 0., 0., 0., 0.],

[0., 0., 1., 0., 0., 0., 0., 0., 0., 0.],

[0., 0., 0., 1., 0., 0., 0., 0., 0., 0.],

[0., 0., 0., 0., 1., 0., 0., 0., 0., 0.],

[0., 0., 0., 0., 0., 1., 0., 0., 0., 0.],

[0., 0., 0., 0., 0., 0., 1., 0., 0., 0.],

[0., 0., 0., 0., 0., 0., 0., 1., 0., 0.],

[0., 0., 0., 0., 0., 0., 0., 0., 1., 0.],

[0., 0., 0., 0., 0., 0., 0., 0., 0., 1.]], dtype = float32)

>>> from keras.utils import normalize

>>> normalize([1, 2, 3, 4, 5])

array([[0.13483997, 0.26967994, 0.40451992, 0.53935989, 0.67419986]])

* **print\_summary**

Có chức năng in ra tổng quan về mô hình.

from keras.utils **import** print\_summary print\_summary(model)

* **plot model**

Được sử dụng để tạo biểu diễn mô hình ở định dạng dấu chấm và lưu vào tệp.

**from** keras.utils **import** plot\_model

plot\_model(model,to\_file = 'image.png')

**Plot Model** này sẽ tạo ra một hình ảnh để hiểu hiệu suất mô hình