Tensorflow:

1. Cách Cài Đặt?

pip3 install tensorflow

1. Cách Import?

import tensorflow as tf

1. Tensor Trong Tensorflow?

* Bản chất là Numpy Array được bọc trong Container

1. Cách Xài GPU?

* Mặc định Tensorflow sẽ xài GPU nếu thấy trong máy có GPU phù hợp, còn không thì CPU

1. Tạo 1 Constant (Eager) Tensor?

<Tensor> = tf.constant(<Mảng>, shape = <Shape>)

* Mặc định

|  |  |
| --- | --- |
| <Shape> | None |

* <Tensor> = <Mảng> được Reshape thành <Shape> nếu <Shape> được chỉ định
* Ví dụ

foo = tf.constant([[1, 2], [4, 5]], shape = (4, ))

* Ta có

|  |  |
| --- | --- |
| foo | [1, 2, 4, 5] |

1. Tạo 1 Variable Tensor?

<Tensor> = tf.constant(<Mảng>)

* Đéo khác gì Tensor cố định nhưng không được Reshape, và có thể thay đổi giá trị In Place
* Để thay đổi giá trị của <Tensor>

<Tensor>.assign(<Giá Trị Mới>)

* <Giá Trị Mới> phải cùng Shape với <Mảng>
* Để + thêm vào <Tensor>

<Tensor>.assign\_add(<Giá Trị Cộng Thêm>)

* <Giá Trị Cộng Thêm> phải cùng Shape với <Mảng>

1. Trả Về Tensor Chứa Toàn 0?

tf.zeros(<Shape>)

* Ví dụ

foo = tf.zeros((2, 3))

1. Trả Về Tensor Nén Hoặc Tích Ngoài?

tf.tensordot(<Tensor 1>, <Tensor 2>, <Chiều>)

* Nếu <Chiều> = 0, thì trả về tích ngoài của <Tensor 1> với <Tensor 2>
* Nếu <Chiều> là 1 Iterable thì nó phải chứa 2 phần tử xác định chiều nén tương ứng của <Tensor 1> và <Tensor 2>
* Ví dụ

foo = tf.tensordot(bar, boo, [2, 4])

1. Gradient?

* Đéo khác gì sơ đồ tính toán trong Torch
* Để bắt đầu Tracking các phép tính, đặt chúng vào Context Manager sau

with tf.GradientTape(<Không Hủy Sơ Đồ>) as <Sơ Đồ Tính Toán>:

<Tính Toán Gì Đó>

* Mặc định

|  |  |
| --- | --- |
| <Không Hủy Sơ Đồ> | False |

* Các Variable Tensor đặt vào đây tương đương Tensor trong Torch với requires\_grad = True, còn các Constant Variable thì = False
* Lưu ý khi bạn tạo ra 1 Tensor mới = biểu thức của các Variable Tensor thì vẫn ra Constant Tensor, nhưng Constant Tensor này vẫn nằm trong sơ đồ tính toán và vẫn có thể được tính Gradient
* Để biến 1 Constant Tensor thành 1 Variable Tensor tạm thời trong Context Manager này

<Sơ Đồ Tính Toán>.watch(<Constant Tensor>)

* Để lan truyền ngược và tính Gradient của <X> ứng với <Loss>

<Gradient> = <Sơ Đồ Tính Toán>.gradient(<Loss>, <X>)

* Sau đó, bạn không thể tính Gradient được với <Sơ Đồ Tính Toán> nữa, nó đã tiêu biến
* Để <Sơ Đồ Tính Toán> không tiêu biến và cho phép bạn tiếp tục tính Gradient, đặt <Không Hủy Sơ Đồ> = True

Tensor:

1. Dấu “@” Có Ý Nghĩa Gì?

* Giống y chang trong Torch, chỉ có điều không tính được tích vô hướng của 2 Vector

Random – Ngẫu Nhiên:

1. Cách Import?

from tensorflow import random

1. Trả Về Tensor Có Giá Trị Thuộc Phân Phối Chuẩn?

random.normal(<Shape>, <Mean>, <Độ Lệch Chuẩn>)

* Mặc định

|  |  |
| --- | --- |
| <Mean> | 0 |
| <Độ Lệch Chuẩn> | 1 |

* Ví dụ

foo = random.normal((3, 4), 4, 5)

Keras:

1. Cách Import?

from tensorflow import keras

1. Keras Tensor?

* Bản chất cũng như Tensor thường nhưng không mang giá trị, có thể Pass cho các lớp
* Tạo 1 Keras Tensor

<Keras Tensor> = keras.Input(<Shape>, <Kích Thước Batch>, <Tên>)

* Mặc định

|  |  |
| --- | --- |
| <Kích Thước Batch> | None |
| <Keras Tensor> Shape | (<Kích Thước Batch>, <Shape>) |

* Lưu ý <Kích Thước Batch> ở đây không có tác dụng gì cả trong quá trình Train hay Test, nhưng nó sẽ là cái được hiển thị khi bạn xem cấu trúc Model
* Ví dụ

foo = keras.Input((12, 15), 20)

* Ta có

|  |  |
| --- | --- |
| foo Shape | (20, 12, 15) |

1. Model?

* Tạo 1 Model

<Model> = keras.Model(<Các Input>, <Các Output>, name = <Tên>)

* <Các Input> là Keras Tensor hoặc là dãy các Keras Tensor
* <Các Output> tương tự, nhưng phải chứa các Keras Tensor xuất phát từ

<Các Input> thông qua quá trình Pass qua các lớp

* Chỉ định Loss và Optimizer cho <Model>

<Model>.compile(<Optimizer>, <Loss>, <In Gì Ra>)

* Train <Model>

<Model>.fit(

<Các Input Tensor>, <Các Output Tensor>, <Kích Thước Batch>,

<Số Epoch>, <Cách Hiển Thị>

)

* Mặc định

|  |  |  |  |
| --- | --- | --- | --- |
| <Kích Thước Batch> | 32 | <Số Epoch> | 1 |
| <Cách Hiển Thị> | 1 |  |  |

* <Các Input Tensor> y chang <Các Input>, chỉ có điều chúng là Tensor thường, tương tự với <Các Output Tensor>
* Các Tensor trong <Các Input Tensor> hay <Các Output Tensor> phải cùng số mẫu, nghĩa là chiều đầu tiên phải = nhau
* Sau mỗi Epoch, Loss và các thứ trong <In Gì Ra> sẽ được in ra
* <In Gì Ra> là 1 mảng gồm các String, mỗi String xác định 1 thứ để in ra, các phần tử hợp lệ của <In Gì Ra>

|  |  |
| --- | --- |
| "accuracy" | In ra độ chính xác |
|  |  |

* <Cách Hiển Thị> xác định có in ra hay không và in kiểu gì

|  |  |
| --- | --- |
| 0 | Không in gì hết |
| 1 | In ra kèm thanh tiến trình |
| 2 | In ra không kèm thanh tiến trình |

* Test <Model>

<Model>.evaluate(

<Các Input Tensor>, <Các Output Tensor>, <Kích Thước Batch>,

<Cách Hiển Thị>

)

* Y chang như khi Train nhưng không cập nhật Parameter
* Dự đoán thử

<Các Output Tensor> = <Model>.predict(<Các Input Tensor>)

* In ra cấu trúc, số lượng Parameter của Model

<Model>.summary()

1. Sequential Model?

* Tạo 1 Sequential Model

<Sequential Model> = keras.Sequential(<Các Lớp>)

* <Các Lớp> là 1 mảng có các phần tử là lớp, các lớp này phải có <Output> là 1 Tensor duy nhất, nếu không sẽ báo lỗi
* <Sequential Model> y chang Model thường, chỉ có điều nhận 1 <Input> và xuất 1 <Output>, giống lớp Sequential trong Torch
* Thông thường <Các Lớp> sẽ được bắt đầu = 1 Keras Tensor, nếu không, thì <Sequential Model> mới đầu sẽ không xác định cấu trúc và số lượng Parameter, sau lần Feed vào <Input> đầu tiên thì mới xác định và cố định luôn

Loss – Thất Thoát:

1. Cách Import?

from tensorflow.keras import losses

1. SCCE (Sparse Categorical Cross Entropy) Loss?

* Tạo 1 SCCE Loss

<SCCE Loss> = losses.SparseCategoricalCrossentropy(

<Chưa Qua Softmax>, <Index Bỏ Qua>, <Cách Tính Loss>

)

* Mặc định

|  |  |
| --- | --- |
| <Chưa Qua Softmax> | False |
| <Index Bỏ Qua> | None |
| <Cách Tính Loss> | "sum\_over\_batch\_size" |

* Forward

<Output> = <SCCE Loss>(<Input>, <Target>, <Weight>)

* Mặc định

|  |  |
| --- | --- |
| <Weight> | None |

* Bản chất y chang lớp Cross Entropy trong Torch, nhưng <Input> phải là nhãn, có giá trị là số nguyên
* Nếu <Chưa Qua Softmax> = True thì <Target> sẽ được đi qua hàm Softmax trước
* Đối chiếu <Cách Tính Loss> trong Torch

|  |  |
| --- | --- |
| Tensorflow | Torch |
| "sum\_over\_batch\_size" | "mean" |
| "sum" | "sum" |
| "none" | "none" |

1. MSE (Mean Squared Error) Loss?

* Tạo 1 MSE Loss

<MSE Loss> = losses.MeanSquaredError(<Cách Tính Loss>)

* Cách hoạt động y chang lớp Mean Squared Error Loss trong Torch

Optimizer – Tối Ưu Hóa:

1. Cách Import?

from tensorflow.keras import optimizers

1. Adam Optimizer?

* Tạo 1 Adam Optimizer

<Adam Optimizer> = optimizers.Adam(<Learning Rate>)

Regularizer – Tinh Chỉnh:

1. Cách Import?

from tensorflow.keras import regularizers

1. L2 Regularization (Ridge Regression)?

* Tạo 1 L2 Regularizer

<L2 Regularizer> = regularizers.L2(<Cường Độ>)

* Mặc định

|  |  |
| --- | --- |
| <Cường Độ> | 0.01 |

* Tính Penalty

<Penalty> = <L2 Regularizer>(<Input>)

* Ta có công thức

<Penalty> = <Cường Độ> \* Sum(<Input>2)

* Khi Train, <Penalty> sẽ được + vào <Loss>, khi đó do phải giảm <Penalty> nên giá trị của <Input> sẽ không bị quá lớn, tránh quá khớp, <Input> ở đây có thể là Weight, Bias, … của các lớp

1. L1 Regularization (Lasso Regression)?

* Tạo 1 L1 Regularizer

<L Regularizer> = regularizers.L1(<Cường Độ>)

* Mặc định

|  |  |
| --- | --- |
| <Cường Độ> | 0.01 |

* Đéo khác gì L2 Regularizer, chỉ có điều

<Penalty> = <Cường Độ> \* Sum(|<Input>|)

Layer – Lớp:

1. Cách Import?

from tensorflow.keras import layers

1. Lớp Tuyến Tính?

* Y chang lớp tuyến tính trong Torch, nhưng không cần xác định

<Kích Thước Input>, nghĩa là <Input> có Shape bất kì, có thể là Tensor thường hoặc Keras Tensor, <Output> sẽ là Tensor thường hoặc Keras Tensor tùy vào <Input>

* Nếu <Input> là Numpy Array sẽ tự động chuyển thành Tensor thường
* Tạo 1 lớp tuyến tính

<Lớp Tuyến Tính> = layers.Dense(

<Kích Thước Output>, <Hàm Kích Hoạt>, <Có Bias Không>,

kernel\_regularizer = <Weight Regularizer>, name = <Tên>

)

* Mặc định

|  |  |
| --- | --- |
| <Hàm Kích Hoạt> | None |
| <Có Bias Không> | True |
| <Weight Regularizer> | None |

* Các giá trị <Hàm Kích Hoạt>

|  |  |
| --- | --- |
| "relu" | ReLU |
|  |  |

* Forward

<Output> = <Lớp Tuyến Tính>(<Input>)

* <Tên> sẽ được hiển thị khi bạn làm gì đó có liên quan đến <Lớp Tuyến Tính>, là 1 String không chứa dấu cách
* Khi Train, <Weight> sẽ được đưa vào <Weight Regularizer> để tính <Penalty>, sau đó <Penalty> sẽ + vào <Loss> cuối cùng để lan truyền ngược

1. 2D CNN?

* Tạo 1 lớp 2D CNN

<Lớp 2D CNN> = layers.Conv2D(

<Số Kênh Output>, <Kernel Shape>, <Stride Shape>, <Padding Shape>,

<Kênh Đầu Hay Cuối>, <Dilation Shape>, <Số Nhóm>, <Hàm Kích Hoạt>,

<Có Bias Không>, name = <Tên>

)

* Mặc định

|  |  |  |  |
| --- | --- | --- | --- |
| <Stride Shape> | (1, 1) | <Padding Shape> | (0, 0) |
| <Kênh Đầu Hay Cuối> | "channels\_last" | <Dilation Shape> | (1, 1) |
| <Số Nhóm> | 1 | <Hàm Kích Hoạt> | None |
| <Có Bias Không> | True |  |  |

* Hoạt động y chang lớp 2D CNN trong Torch
* Nếu <Kênh Đầu Hay Cuối> = "channels\_last", thì chiều tương ứng với kênh ảnh sẽ ở cuối, ví dụ Shape (32, 32, 3), nếu = "channels\_first" thì ở đầu, ví dụ

(3, 32, 32), dựa theo đó mà Kernel sẽ trượt lên ảnh theo chiều tương ứng

1. 2D Max Pool?

* Tạo 1 lớp 2D Max Pool

<Lớp 2D Max Pool> = layers.MaxPooling2D(

<Kernel Shape>, <Stride Shape>, <Padding Shape>, <Kênh Đầu Hay Cuối>,

name = <Tên>

)

* Mặc định

|  |  |  |  |
| --- | --- | --- | --- |
| <Kernel Shape> | (2, 2) | <Padding Shape> | (0, 0) |
| <Stride Shape> | <Kernel Shape> | <Kênh Đầu Hay Cuối> | "channels\_last" |

* Đéo khác gì lớp 2D Max Pool trong Torch

1. Flatten?

* Tạo 1 lớp Flatten

<Lớp Flatten> = layers.Flatten(<Kênh Đầu Hay Cuối>, name = <Tên>)

* Mặc định

|  |  |
| --- | --- |
| <Kênh Đầu Hay Cuối> | "channels\_last" |

* Đéo khác gì lớp Flatten trong Torch với các tham số mặc định, nhưng nếu

<Kênh Đầu Hay Cuối> = "channels\_first" thì thứ tự đọc giá trị sẽ thay đổi, ưu tiên đọc theo chiều đầu tiên trước

1. Batch Norm?

* Tạo 1 lớp Batch Norm

<Lớp Batch Norm> = layers.BatchNormalization(

<Kênh Thuộc Chiều Nào>, <Momentum>, <Epsilon>, name = <Tên>

)

* Mặc định

|  |  |  |  |
| --- | --- | --- | --- |
| <Kênh Thuộc Chiều Nào> | –1 | <Momentum> | 0.99 |
| <Epsilon> | 1e–3 |  |  |

* Đéo khác gì lớp Batch Norm trong Torch, áp dụng cho cả 1D, 2D, …, chỉ có điều <Momentum> trong đây = 1 – <Momentum> trong Torch
* Ví dụ
* Nếu <Input> có Shape là (1000, 3, 32, 32), chiều của kênh là chiều có kích thước = 3, thì đặt <Kênh Thuộc Chiều Nào> = 1, vì lớp này chuẩn hóa theo chiều của kênh

1. ReLU?

* Tạo 1 lớp ReLU

<Lớp ReLU> = layers.ReLU(

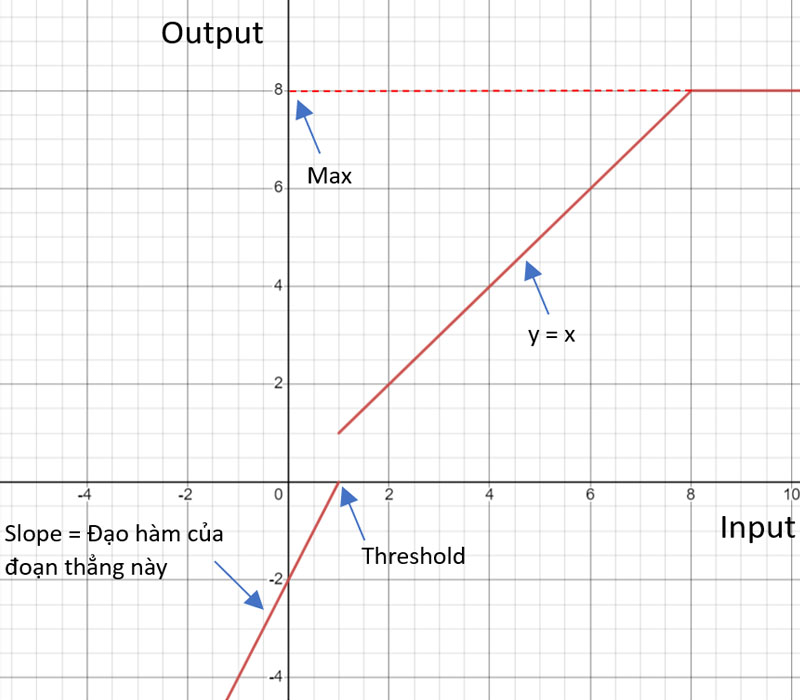
<Max>, <Slope>, <Threshold>, name = <Tên>

)

* Mặc định

|  |  |  |  |
| --- | --- | --- | --- |
| <Max> | None | <Slope> | 0 |
| <Threshold> | 0 |  |  |

* Minh họa



1. Drop Out?

* Tạo 1 lớp Drop Out

<Lớp Drop Out> = layers.Dropout(

<Tỉ Lệ Drop Out>, <Shape Lặp>, name = <Tên>

)

* Mặc định

|  |  |
| --- | --- |
| <Shape Lặp> | None |

* Forward

<Output> = <Lớp Drop Out>(<Input>, <Đang Train À>)

* Mặc định

|  |  |
| --- | --- |
| <Đang Train À> | True |

* Đéo khác gì lớp Drop Out trong Torch, chỉ có điều nếu <Shape Lặp> được xác định thì Tensor chỉ chứa 0 và 1 được tạo ra sẽ có Shape là <Shape Lặp>, sau đó Broadcast với <Input>
* Đặt <Đang Train À> = False để vô hiệu hóa <Lớp Drop Out>, nó tự động vô hiệu hóa khi Test

1. RNN?

* Tạo 1 lớp RNN

<Lớp RNN> = layers.SimpleRNN(

<Số Neuron Hidden>,

return\_sequences = <Trả Về Tất Cả Hidden>,

return\_state = <Trả Về Hidden Cuối>,

name = <Tên>

)

* Mặc định

|  |  |
| --- | --- |
| <Trả Về Tất Cả Hidden> | False |
| <Trả Về Hidden Cuối> | False |

* Forward
* Trường hợp 1, mặc định

<Hidden Cuối> = <Lớp RNN>(<Input>)

* Trường hợp 2, <Trả Về Tất Cả Hidden> = True, <Trả Về Hidden Cuối> = False

<Tất Cả Hidden> = <Lớp RNN>(<Input>)

* Trường hợp 3, <Trả Về Tất Cả Hidden> = True, <Trả Về Hidden Cuối> = True

<Tất Cả Hidden>, <Hidden Cuối> = <Lớp RNN>(<Input>)

* Cơ chế đéo khác gì lớp RNN trong Torch trừ việc chỉ có 1 Stack và không Train được 2 chiều

Data – Dữ Liệu:

1. Cách Import?

from tensor.keras import datasets

1. Bộ Dữ Liệu Nhìn Hình Đoán Số MNIST?

(<X Train>, <Y Train>), (<X Test>, <Y Test>) = datasets.mnist.load\_data()

* Lệnh trên sẽ tải File “mnist.npz” chứa dữ liệu vào thư mục “C:\Users\pc\.keras\datasets”, trong Google Colab thì tải vào thư mục “/root/.keras/datasets”, lưu ý thư mục “.keras” bị ẩn trong Google Colab
* Nếu đã có File “mnist.npz” chứa đúng dữ liệu mong muốn thì không tải
* <X Train> là 60000 tấm ảnh 28 x 28 có giá trị là số nguyên từ 0 đến 255 ứng với <Y Train> là 60000 nhãn có giá trị là số nguyên từ 0 đến 9
* <X Test> và <Y Test> tương tự nhưng chỉ có 10000 tấm ảnh

|  |  |  |  |
| --- | --- | --- | --- |
| <X Train> Shape | <Y Train Shape> | <X Test> Shape | <Y Test Shape> |
| (60000, 28, 28) | (60000) | (10000, 28, 28) | (10000) |

1. Bộ Dữ Liệu Nhìn Hình Màu Đoán Vật CIPHAR 10?

(<X Train>, <Y Train>), (<X Test>, <Y Test>) = datasets.cifar10.load\_data()

* Hoạt động y chang nhự khi tải MNIST
* <X Train> là 50000 tấm ảnh màu RGB 32 x 32, có giá trị nguyên từ 0 đến 255 ứng với <Y Train> là 50000 nhãn có giá trị là số nguyên từ 0 đến 9, tương ứng máy bay, xe hơi, chim, mèo, hươu, chó, ếch, ngựa, tàu, xe tải
* <X Test> và <Y Test> tương tự nhưng chỉ có 10000 tấm ảnh

|  |  |  |  |
| --- | --- | --- | --- |
| <X Train> Shape | <Y Train Shape> | <X Test> Shape | <Y Test Shape> |
| (50000, 32, 32, 3) | (50000, 1) | (10000, 32, 32, 3) | (10000, 1) |