

TRƯỜNG ĐẠI HỌC TRẦN ĐẠI NGHĨA
KHOA CÔNG NGHỆ THÔNG TIN



BÁO CÁO

BÀI TẬP LỚN MÔN HỌC MẠNG NƠI NƠI

Đề tài: “Tìm hiểu ngôn ngữ Python, thư viện Pytorch
và viết ứng dụng phân lớp hình ảnh
(frog, horse, ship, truck)”

TP. HỒ CHÍ MINH, THÁNG 10 NĂM 2019

TRƯỜNG ĐẠI HỌC TRẦN ĐẠI NGHĨA
KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO

**BÀI TẬP LỚN MÔN HỌC
MẠNG NƠI NƠI**

Đề tài: “Tìm hiểu ngôn ngữ Python, thư viện Pytorch
và viết ứng dụng phân lớp hình ảnh
(frog, horse, ship, truck)”

GVHD: Ngô Thanh Tú

Sinh viên thực hiện:

Nguyễn Lê Xuân Phước

Tôn Nữ Nguyên Hậu

Đỗ Tống Quốc

Lớp: 16DDS06031

TP. HỒ CHÍ MINH, THÁNG 10 NĂM 2019

MỤC LỤC

DANH MỤC HÌNH ẢNH.....	6
LỜI MỞ ĐẦU	7
CHƯƠNG 1. TÌM HIỂU NGÔN NGỮ LẬP TRÌNH PYTHON	8
 1.1. Giới thiệu ngôn ngữ Python	8
1.1.1. Lịch sử phát triển	8
1.1.2. Phiên bản của Python.....	10
1.1.3. Một điểm khác nhau giữa phiên bản 3.x và 2.x.....	10
1.1.4. Đặc điểm của Python	16
 1.2. Hướng dẫn cài đặt Ananconda	25
1.2.1. Giới thiệu Ananconda	25
1.2.2. Download Anaconda và hướng dẫn cài đặt trên HĐH Window	26
1.2.3. Hướng dẫn cài đặt thêm thư viện	31
 1.3. Hướng dẫn cài đặt IDE Spyder	32
1.3.1. Giới thiệu IDE Spyder.....	32
1.3.2. Hướng dẫn cài đặt IDE Spyder bằng Navigator	33
1.3.4. Hướng dẫn sử dụng Spyder.....	35
CHƯƠNG 2. GIỚI THIỆU VỀ MẠNG NƠRON	37
 2.1. Giới thiệu về Maching Learning.....	37
 2.2. Lịch sử phát triển mạng nơron nhân tạo- ANN	42
 2.3. Lịch sử phát triển Deep Learing	44
 2.4. Một số thư viện Deep Learning nổi tiếng Hiện nay	49
 2.5. Các khái niệm trong mạng Noron	53
2.5.1. Epoch.....	53
2.5.2. Batch Size.....	54
2.5.3. Iterations.....	54
2.5.4. Loss Fuction	54
2.5.5. Momentum	54
2.5.6. Dataset.....	54
2.5.7. Learning rate	54
2.5.8. Traing set.....	55
2.5.9. Testing set	55
CHƯƠNG 3. GIỚI THIỆU VỀ THU VIỆN PYTORCH	56

3.1.	<i>Lịch sử phát triển và các phiên bản Pytorch.....</i>	56
3.2.	<i>Đặc điểm của Pytorch.....</i>	56
3.3.	<i>Cài đặt thư viện Pytorch.....</i>	57
3.4.	<i>Tổng quan về hướng dẫn sử dụng thư viện Pytorch</i>	58
3.4.1.	Mở đầu (Getting Started)	59
3.4.2.	Ảnh (Image)	62
3.4.3.	Âm thanh (Audio)	64
3.4.4.	Văn bản (Text)	64
3.4.5.	Generative	65
3.4.6.	Học tăng cường (Reinforcement Learning)	66
3.4.7.	Mở rộng Pytorch	66
3.4.8.	Sử dụng và sản xuất	67
3.4.9.	Pytorch trong các ngôn ngữ khác.....	69
3.5.	<i>Deep Learning Với Pytorch: Một Blitz 60 Phút.....</i>	70
3.5.1.	Pytorch là gì?	70
3.5.2.	Autograd: Automatic Differentiation (Autograd: Tính đạo hàm tự động)	76
3.5.3.	Mạng nơron	78
3.5.4.	Huấn Luyện Một Phân Loại	83
CHƯƠNG 4. XÂY DỰNG ỨNG DỤNG.....		91
4.1.	<i>Bài toán xây dựng bộ phân lớp hình ảnh</i>	91
4.2.	<i>Giới thiệu tập dữ liệu CIFAR10 và trình bày các bước đã thực hiện để lấy tập dữ liệu từ tập CIFAR10</i>	92
4.2.1.	Giới thiệu tập dữ liệu CIFAR10	92
4.2.2.	Các bước đã thực hiện để lấy tập dữ liệu từ tập CIFAR10	93
4.3.	<i>Phương pháp lựa chọn của đè tài để xây dựng bộ phân lớp hình ảnh (Mạng nơron tích chập – DCNN (Deep Convolution Neural Network)).....</i>	93
4.4.	<i>Một số lớp cơ bản của 1 DCNN được sử dụng trong ứng dụng (Deep Convolution Neural Network).....</i>	95
4.4.1.	Lớp tích chập:.....	95
4.4.2.	Lớp pooling:	97
4.4.3.	Hàm truyền (activation function):	98
4.4.4.	Lớp dropout:	99

4.4.5. Lớp liên kết đầy đủ (fully connected layer):	99
4.4. Giao diện, các chức năng của ứng dụng xây dựng bộ phân lớp hình ảnh bằng mạng DCNN	100
KẾT LUẬN	104
TÀI LIỆU THAM KHẢO	105

DANH MỤC HÌNH ẢNH

<i>Hình 1. 1: Guido van Russom</i>	8
<i>Hình 1. 2: Giao diện trang chủ Anaconda.....</i>	28
<i>Hình 1. 3: Trang tải Anaconda</i>	28
<i>Hình 1. 4: Chạy chương trình vừa tải về</i>	27
<i>Hình 1. 5: License Agreement.....</i>	27
<i>Hình 1. 6: Chọn loại cài đặt.....</i>	28
<i>Hình 1. 7: Chọn Vị trí cài đặt</i>	28
<i>Hình 1. 8: Advanced Options.....</i>	29
<i>Hình 1. 9: Hoàn thành cài đặt</i>	29
<i>Hình 1. 10: Anaconda và JetBrains</i>	30
<i>Hình 1. 11: Cài đặt hoàn tất và khởi động chương trình</i>	30
<i>Hình 1. 12: Giao diện chính của IDE Spyder.....</i>	33
<i>Hình 1. 13: Chạy Spyder.....</i>	34
<i>Hình 1. 14: Cài đặt Spyder bằng Anaconda Navigator.....</i>	34
<i>Hình 1. 15: Mở chương trình Spyder.....</i>	35
<i>Hình 1. 16: Giao diện chính Spyder</i>	35
<i>Hình 1. 17: Lưu chương trình với đuôi *.py</i>	38
<i>Hình 1. 18: Chạy chương trình</i>	38
<i>Hình 2. 1: Đường thẳng trên mặt phẳng để phân chia hai tập điểm.....</i>	39
<i>Hình 2. 2: Đường thẳng phân chia không tồn tại</i>	40
<i>Hình 2. 3: Lịch sử phát triển của Machine Learning</i>	42
<i>Hình 2. 4: Lịch sử phát triển Deep Learning.....</i>	44
<i>Hình 2. 5: Kết quả ILSVRC qua các năm</i>	48
<i>Hình 2. 6: Một số thư viện Deep learning nổi tiếng</i>	49
<i>Hình 3. 1: Trang tải thư viện Pytorch.....</i>	57
<i>Hình 3. 2: Chương trình chữ số viết tay mnist.....</i>	58
<i>Hình 3. 3: Trang https://pytorch.org/tutorials/.....</i>	59
<i>Hình 3. 4: Mạng này phân loại các ảnh số.....</i>	78
<i>Hình 3. 5: Tập dữ liệu CIFAR10.....</i>	84
<i>Hình 4. 1: Tập dữ liệu CIFAR10.....</i>	93
<i>Hình 4. 2: Hàm truyền</i>	98
<i>Hình 4. 3: Giao diện chính chương trình.....</i>	100
<i>Hình 4. 4: Load và chuẩn hóa các tập dữ liệu huấn luyện/ kiểm tra</i>	100
<i>Hình 4. 5: Mô hình mạng nơron</i>	101
<i>Hình 4. 6: Hàm mất mát và tối ưu hóa</i>	101
<i>Hình 4. 7: Kết quả Huấn luyện mạng trên tập dữ liệu huấn luyện.....</i>	102
<i>Hình 4. 8: Kết quả Kiểm tra mạng trên tập dữ liệu kiểm tra.....</i>	102

LỜI MỞ ĐẦU

. Theo sự phát triển của thế giới, công nghệ nhận dạng hình ảnh là một công nghệ mới giúp máy tính nhận dạng được các hình ảnh qua các tập dữ liệu các hình ảnh thường được sử dụng để đào tạo các thuật toán máy học. Nó là một trong những bộ dữ liệu được sử dụng rộng rãi nhất cho nghiên cứu máy học. Bộ dữ liệu CIFAR-10 chứa 60.000 hình ảnh màu 32x32 trong 10 lớp khác nhau. 10 lớp khác nhau đại diện cho máy bay, ô tô, chim, mèo, hươu, chó, éch, ngựa, tàu và xe tải. Có 6.000 hình ảnh của mỗi lớp.

Mạng nơron di truyền có thể học kiến thức thông qua một số lượng lớn các mẫu dạy để thu được khả năng dự báo, và quá trình xử lý có thể rút ra hoàn toàn mà không nhấn mạnh giả thuyết dựa trên sự phân bố của dữ liệu và không cần thiết giải thích chi tiết của kiến thức. Mạng nơron có thể tự động không chê tất cả các mối quan hệ bên trong của hệ thống theo các mẫu hiện thời (được tạo sẵn).

Em xin chân thành cảm ơn thầy **Ngô Thanh Tú** đã tận tình hướng dẫn và giúp đỡ em hoàn thành cuốn đồ án này.

Do kinh nghiệm và kiến thức chưa được sâu sắc nên trong báo cáo về đề tài của nhóm mong thầy góp ý thêm để nhóm có thể hoàn thiện tốt hơn các đề tài nghiên cứu về sau.

Chúng em xin chân thành cảm ơn !

TP. Hồ Chí Minh, ngày 05 tháng 10 năm 2019

CHƯƠNG 1. TÌM HIỂU NGÔN NGỮ LẬP TRÌNH PYTHON

1.1. Giới thiệu ngôn ngữ Python

Python là một ngôn ngữ lập trình thông dịch do Guido van Rossum tạo ra năm 1990. Python hoàn toàn tạo kiểu động và dùng cơ chế cấp phát bộ nhớ tự động; do vậy nó tương tự như Perl, Ruby, Scheme, Smalltalk, và Tcl. Python được phát triển trong một dự án mã mở, do tổ chức phi lợi nhuận Python Software Foundation quản lý.

Theo đánh giá của Eric S. Raymond, Python là ngôn ngữ có hình thức rất sáng sủa, cấu trúc rõ ràng, thuận tiện cho người mới học lập trình. Cấu trúc của Python còn cho phép người sử dụng viết mã lệnh với số lần gõ phím tối thiểu, như nhận định của chính Guido van Rossum trong một bài phỏng vấn ông.

Ban đầu, Python được phát triển để chạy trên nền Unix. Nhưng rồi theo thời gian, nó đã “bành trướng” sang mọi hệ điều hành từ MS-DOS đến Mac OS, OS/2, Windows, Linux và các hệ điều hành khác thuộc họ Unix. Mặc dù sự phát triển của Python có sự đóng góp của rất nhiều cá nhân, nhưng Guido van Rossum hiện nay vẫn là tác giả chủ yếu của Python. Ông giữ vai trò chủ chốt trong việc quyết định hướng phát triển của Python.

1.1.1. Lịch sử phát triển

Python được hình thành vào cuối những năm 1980, và việc thực hiện nó vào tháng 12 năm 1989 bởi Guido van Rossum tại Centrum Wiskunde & Informatica (CWI) ở Hà Lan như là một kế thừa cho ngôn ngữ ABC (tự lấy cảm hứng từ SETL) có khả năng xử lý ngoại lệ và giao tiếp với Hệ điều hành Amoeba. Van Rossum là tác giả chính của Python, và vai trò trung tâm của ông trong việc quyết định hướng phát triển của Python.



Hình 1. 1: Guido van Rossum

Sự phát triển Python đến nay có thể chia làm các giai đoạn:

Python 1: bao gồm các bản phát hành 1.x. Giai đoạn này, kéo dài từ đầu đến cuối thập niên 1990. Từ năm 1990 đến 1995, Guido làm việc tại CWI (Centrum voor Wiskunde en Informatica - Trung tâm Toán-Tin học tại Amsterdam, Hà Lan). Vì vậy, các phiên bản Python đầu tiên đều do CWI phát hành. Phiên bản cuối cùng phát hành tại CWI là 1.2.

Vào năm 1995, Guido chuyển sang CNRI (Corporation for National Research Initiatives) ở Reston, Virginia. Tại đây, ông phát hành một số phiên bản khác. Python 1.6 là phiên bản cuối cùng phát hành tại CNRI.

Sau bản phát hành 1.6, Guido rời bỏ CNRI để làm việc với các lập trình viên chuyên viết phần mềm thương mại. Tại đây, ông có ý tưởng sử dụng Python với các phần mềm tuân theo chuẩn GPL. Sau đó, CNRI và FSF (Free Software Foundation - Tổ chức phần mềm tự do) đã cùng nhau hợp tác để làm bản quyền Python phù hợp với GPL. Cùng năm đó, Guido được nhận Giải thưởng FSF vì Sự phát triển Phần mềm tự do (Award for the Advancement of Free Software).

Phiên bản 1.6.1 ra đời sau đó là phiên bản đầu tiên tuân theo bản quyền GPL. Tuy nhiên, bản này hoàn toàn giống bản 1.6, trừ một số sửa lỗi cần thiết.

Python 2: vào năm 2000, Guido và nhóm phát triển Python dời đến BeOpen.com và thành lập BeOpen PythonLabs team. Phiên bản Python 2.0 được phát hành tại đây. Sau khi phát hành Python 2.0, Guido và các thành viên PythonLabs gia nhập Digital Creations.

Python 2.1 ra đời kế thừa từ Python 1.6.1 và Python 2.0. Bản quyền của phiên bản này được đổi thành Python Software Foundation License. Từ thời điểm này trở đi, Python thuộc sở hữu của Python Software Foundation (PSF), một tổ chức phi lợi nhuận được thành lập theo mẫu Apache Software Foundation.

Python 3, còn gọi là Python 3000 hoặc Py3K: Dòng 3.x sẽ không hoàn toàn tương thích với dòng 2.x, tuy vậy có công cụ hỗ trợ chuyển đổi từ các phiên bản 2.x sang 3.x. Nguyên tắc chủ đạo để phát triển Python 3.x là "bỏ cách làm việc cũ nhằm hạn chế trùng lặp về mặt chức năng của Python". Trong PEP (Python Enhancement Proposal) có mô tả chi tiết các thay đổi trong Python. Các đặc điểm mới của Python 3.0 sẽ được trình bày phần cuối bài này.

1.1.2. Phiên bản của Python

- *CPython*

Đây là phiên bản đầu tiên và được duy trì lâu nhất của Python, được viết trong C. Những đặc điểm của ngôn ngữ mới xuất hiện đầu tiên từ đây

- *Jython*

Là phiên bản Python trong môi trường Java. Bản này có thể được sử dụng như là một ngôn ngữ script cho những ứng dụng Java. Nó cũng thường được sử dụng để tạo ra những tests thử nghiệm cho thư viện Java

- *Python for .NET*

Phiên bản này thật ra sử dụng phiên bản Cpython, nhưng nó là một ứng dụng .NET được quản lý, và tạo ra thư viện .NET sẵn dùng. Bản này được xây dựng bởi Brian Lloyd.

- *IronPython*

Là một bản Python tiếp theo của .NET, không giống như Python.NET đây là một bản Python hoàn chỉnh, tạo ra IL, và biên dịch mã Python trực tiếp vào một tập hợp .NET.

- *PyPy*

PyPy được viết trên Python, thậm chí cả việc thông dịch từng byte mã cũng được viết trên Python. Nó sử dụng Cpython như một trình thông dịch cơ sở. Một trong những mục đích của dự án cho ra đời phiên bản này là để khuyến khích sự thử nghiệm bản thân ngôn ngữ này làm cho nó dễ dàng hơn để sửa đổi thông dịch (bởi vì nó được viết trên Python).

1.1.3. Một điểm khác nhau giữa phiên bản 3.x và 2.x

- *Division operator*

Thay đổi này đặc biệt nguy hiểm nếu bạn đang thực thi code Python 3 trong Python 2, vì thay đổi trong hành vi phân chia số nguyên thường có thể không được chú ý (nó không làm tăng cú pháp SyntaxError).

```
print 7 / 5
print -7 / 5
Output in Python 2.x:
1
-2
Output in Python 3.x:
1.4
-1.4
```

- *Print function*

Đây là một trong những sự thay đổi được biết đến nhiều nhất từ bản Python 2.x lên Python 3.x. Python 2.x không có vấn đề với các lệnh bổ sung, nhưng ngược lại, Python 3.x sẽ tăng cú pháp SyntaxError nếu chúng ta gọi hàm in mà không có dấu ngoặc đơn.

```
print 'Hello, Geeks'      # Python 3.x doesn't support
print('Hope You like these facts')
Output in Python 2.x:
Hello, Geeks
Hope You like these facts

Output in Python 3.x:
File "a.py", line 1
    print 'Hello, Geeks'
          ^
SyntaxError: invalid syntax
```

- *Unicode*

Trong Python 2.x, kiểu mặc định của String là ASCII, nhưng ở Python 3.x kiểu mặc định của String là Unicode và 2 lớp byte: byte và bytearrays.

```
print(type('default string '))
print(type(u'string with b '))
Output in Python 2.x (Unicode and str are different):
<type 'str'>
<type 'unicode'>

Output in Python 3.x (Unicode and str are same):
<class 'str'>
<class 'str'>
```

- *Xrange*

Việc sử dụng xrange () rất phổ biến trong Python 2.x để tạo một đối tượng có thể lặp lại, ví dụ: trong vòng lặp for hoặc list / set-dictionary-comprehension. xrange-iterable không phải hoàn toàn có nghĩa là bạn có thể lặp lại nó vô hạn.

Trong Python 3, range () đã được thực hiện giống như hàm xrange () sao cho hàm xrange () chuyên dụng không còn tồn tại nữa (xrange () tăng một NameError trong Python 3).

```
for x in xrange(1, 5):
    print(x),

for x in range(1, 5):
    print(x),
Output in Python 2.x:
1 2 3 4 1 2 3 4
Output in Python 3.x:
```

```
NameError: name 'xrange' is not defined
```

- *Error Handling*

Đây là một thay đổi nhỏ trên phiên bản 3.x, từ khoá as đã trở thành bắt buộc Kiểm tra không có từ khoá as trong 2 phiên bản Python

```
try:  
    trying_to_check_error  
except NameError, err:  
    print err, 'Error Caused' # Would not work in Python 3.x  
Output in Python 2.x:  
name 'trying_to_check_error' is not defined Error Caused  
  
Output in Python 3.x :  
File "a.py", line 3  
    except NameError, err:  
           ^  
SyntaxError: invalid syntax
```

Kiểm tra khi có từ khoá as trong 2 phiên bản Python

```
try:  
    trying_to_check_error  
except NameError as err: # 'as' is needed in Python 3.x  
    print (err, 'Error Caused')  
Output in Python 2.x:  
(NameError("name 'trying_to_check_error' is not defined",), 'Error Caused')  
  
Output in Python 3.x:  
name 'trying_to_check_error' is not defined Error Caused
```

- *The next() function and next() method*

Next() hay (.next ()) là một hàm thường được sử dụng (method), đây là một thay đổi cú pháp khác (hay đúng hơn là thay đổi trong thực thi) sử dụng tốt trong python 2 nhưng bị loại bỏ trong python3

Vd: python 2.x:

```
my_generator = (letter for letter in 'abcdefg')  
  
next(my_generator)  
my_generator.next()  
-----  
result:  
'a'  
'b'
```

python3.x:

```
my_generator = (letter for letter in 'abcdefg')
next(my_generator)

-----
'a'

my_generator.next()

-----
AttributeError           Traceback (most recent call last)

<ipython-input-14-125f388bb61b> in <module>()
----> 1 my_generator.next()

AttributeError: 'generator' object has no attribute 'next'
```

- *Các biến vòng lặp và rò rỉ namespace chung*

Trong Python 3.x cho các biến vòng lặp không bị rò rỉ vào không gian tên chung nữa List comprehensions không còn hỗ trợ mẫu cú pháp [... for var in item1, item2, ...]. Sử dụng [... for var in (item1, item2, ...)] thay thế. Cũng lưu ý rằng việc hiểu danh sách có ngữ nghĩa khác nhau: chúng gần với cú pháp cho biểu thức máy hiểu bên trong một hàm tạo danh sách (và đặc biệt là các biến điều khiển vòng lặp không còn bị rò rỉ ra ngoài phạm vi.”

Vd: Python 2.x

```
i = 1
print 'before: i =', i
print 'comprehension: ', [i for i in range(5)]

print 'after: i =', i
-----
before: i = 1
comprehension:  [0, 1, 2, 3, 4]
after: i = 4
```

Python 3.x:

```
i = 1
print('before: i =', i)

print('comprehension:', [i for i in range(5)])
print('after: i =', i)
-----
before: i = 1
comprehension: [0, 1, 2, 3, 4]
after: i = 1
```

- *So sánh khác loại*

Ở python 2 , có thể so sánh list với string,... nhưng ở python 3 thì không
Vd: python 2.x

```
print "[1, 2] > 'foo' = ", [1, 2] > 'foo'
print "(1, 2) > 'foo' = ", (1, 2) > 'foo'
print "[1, 2] > (1, 2) = ", [1, 2] > (1, 2)
-----
[1, 2] > 'foo' = False
(1, 2) > 'foo' = True
[1, 2] > (1, 2) = False
```

Python 3.x

```
print("[1, 2] > 'foo' = ", [1, 2] > 'foo')
print("(1, 2) > 'foo' = ", (1, 2) > 'foo')
print("[1, 2] > (1, 2) = ", [1, 2] > (1, 2))
-----
TypeError Traceback (most recent call last)

<ipython-input-16-a9031729f4a0> in <module>()
      1 print('Python', python_version())
----> 2 print("[1, 2] > 'foo' = ", [1, 2] > 'foo')
      3 print("(1, 2) > 'foo' = ", (1, 2) > 'foo')
      4 print("[1, 2] > (1, 2) = ", [1, 2] > (1, 2))
TypeError: unorderable types: list() > str()
```

- *Phân tích cú pháp đầu vào của người dùng thông qua input()*

Hàm input () được cố định trong Python 3 để nó luôn lưu trữ đầu vào của người dùng làm các đối tượng str. Để tránh hành vi nguy hiểm trong Python 2 để đọc trong các loại khác so với chuỗi, chúng ta phải sử dụng raw_input () để thay thế.

Vd: Python 2.x

```
Python 2.7.6
[GCC 4.0.1 (Apple Inc. build 5493)] on darwin
Type "help", "copyright", "credits" or "license" for more information.

>>> my_input = input('enter a number: ')
enter a number: 123

>>> type(my_input)
<type 'int'>

>>> my_input = raw_input('enter a number: ')
enter a number: 123

>>> type(my_input)
```

```
<type 'str'>
```

Python 3.x

```
Python 3.4.1
[GCC 4.2.1 (Apple Inc. build 5577)] on darwin
Type "help", "copyright", "credits" or "license" for more information.

>>> my_input = input('enter a number: ')
enter a number: 123

>>> type(my_input)
<class 'str'>
```

Trả về các đối tượng có thể lặp lại thay vì danh sách

- Một số hàm và phương thức trả về các đối tượng có thể lặp lại trong Python 3 ngay-thay vì các danh sách trong Python 2

Vd: Python 2.x

```
print range(3)
print type(range(3))
-----
[0, 1, 2]
<type 'list'>
```

Python 3

```
print(range(3))
print(type(range(3)))
print(list(range(3)))
-----
range(0, 3)
<class 'range'>
[0, 1, 2]
```

Một số hàm và phương thức thường được sử dụng không trả về danh sách nữa trong Python 3:

```
zip()
map()
filter()
dictionary's .keys() method
dictionary's .values() method
```

```
dictionary's .items() method
```

1.1.4. Đặc điểm của Python

- Dễ học, dễ đọc

Python được thiết kế để trở thành một ngôn ngữ dễ học, mã nguồn dễ đọc, bô cục trực quan, dễ hiểu, thể hiện qua các điểm sau:

- Từ khóa

- o Python tăng cường sử dụng từ khóa tiếng Anh, hạn chế các kí hiệu và cấu trúc cú pháp so với các ngôn ngữ khác.
- o Python là một ngôn ngữ phân biệt kiểu chữ HOA, chữ thường.
- o Như C/C++, các từ khóa của Python đều ở dạng chữ thường.

- Khối lệnh

Trong các ngôn ngữ khác, khối lệnh thường được đánh dấu bằng cặp kí hiệu hoặc từ khóa. Ví dụ, trong C/C++, cặp ngoặc nhọn { } được dùng để bao bọc một khối lệnh. Python, trái lại, có một cách rất đặc biệt để tạo khối lệnh, đó là thụt các câu lệnh trong khối vào sâu hơn (về bên phải) so với các câu lệnh của khối lệnh cha chứa nó. Ví dụ, giả sử có đoạn mã sau trong C/C++:

```
#include <math.h>
//...
delta = b * b - 4 * a * c;
if (delta > 0) {
    // Khoi lenh moi bat dau tu kí tu { den }
    x1 = (- b + sqrt(delta)) / (2 * a);
    x2 = (- b - sqrt(delta)) / (2 * a);
    printf("Phuong trinh co hai nghiem phan biet:\n");
    printf("x1 = %f; x2 = %f", x1, x2);
}
```

Đoạn mã trên có thể được viết lại bằng Python như sau:

```
# -*- coding: utf-8 -*-
"""
Spyder Editor
This is a temporary script file.
"""

import math
#...
```

```

delta = b * b - 4 * a * c
if delta > 0:
    # Khối lệnh mới, thụt vào đầu dòng
    x1 = (- b + math.sqrt(delta)) / (2 * a)
    x2 = (- b - math.sqrt(delta)) / (2 * a)
    print "Phuong trinh co hai nghiem phan biet:"
    print "x1 = ", x1, "; ", "x2 = ", x2

```

Ta có thể sử dụng dấu tab hoặc khoảng trắng để thụt các câu lệnh vào.

- Các bản hiện thực

Python được viết từ những ngôn ngữ khác, tạo ra những bản hiện thực khác nhau. Bản hiện thực Python chính, còn gọi là CPython, được viết bằng C, và được phân phối kèm một thư viện chuẩn lớn được viết hỗn hợp bằng C và Python. CPython có thể chạy trên nhiều nền và khả chuyển trên nhiều nền khác. Dưới đây là các nền trên đó, CPython có thể chạy.

- Các hệ điều hành họ Unix: AIX, Darwin, FreeBSD, Mac OS X, NetBSD, Linux, OpenBSD, Solaris,...
- Các hệ điều hành dành cho máy desktop: Amiga, AROS, BeOS, Mac OS 9, Microsoft Windows, OS/2, RISC OS.
- Các hệ thống nhúng và các hệ đặc biệt: GP2X, Máy ảo Java, Nokia 770 Internet Tablet, Palm OS, PlayStation 2, PlayStation Portable, Psion, QNX, Sharp Zaurus, Symbian OS, Windows CE/Pocket PC, Xbox/XBMC, VxWorks.
- Các hệ máy tính lớn và các hệ khác: AS/400, OS/390, Plan 9 from Bell Labs, VMS, z/OS.

Ngoài CPython, còn có hai hiện thực Python khác: Jython cho môi trường Java và IronPython cho môi trường .NET và Mono.

- Khả năng mở rộng

Python có thể được mở rộng: nếu ta biết sử dụng C, ta có thể dễ dàng viết và tích hợp vào Python nhiều hàm tùy theo nhu cầu. Các hàm này sẽ trở thành hàm xây dựng sẵn (built-in) của Python. Ta cũng có thể mở rộng chức năng của trình thông dịch, hoặc liên kết các chương trình Python với các thư viện chỉ ở dạng nhị phân (như các thư viện đồ họa do nhà sản xuất thiết bị cung cấp). Hơn thế nữa, ta cũng có thể liên kết trình thông dịch của Python với các ứng dụng viết từ C và sử dụng nó như là một mở rộng hoặc một ngôn ngữ dòng lệnh phụ trợ cho ứng dụng đó.

- Trình thông dịch

Python là một ngôn ngữ lập trình dạng thông dịch, do đó có ưu điểm tiết kiệm thời gian phát triển ứng dụng vì không cần phải thực hiện biên dịch

và liên kết. Trình thông dịch có thể được sử dụng để chạy file script, hoặc cũng có thể được sử dụng theo cách tương tác. Ở chế độ tương tác, trình thông dịch Python tương tự shell của các hệ điều hành họ Unix, tại đó, ta có thể nhập vào từng biểu thức rồi gõ Enter, và kết quả thực thi sẽ được hiển thị ngay lập tức. Đặc điểm này rất hữu ích cho người mới học, giúp họ nghiên cứu tính năng của ngôn ngữ; hoặc để các lập trình viên chạy thử mã lệnh trong suốt quá trình phát triển phần mềm. Ngoài ra, cũng có thể tận dụng đặc điểm này để thực hiện các phép tính như với máy tính bỏ túi.

- Lệnh và cấu trúc điều khiển

Mỗi câu lệnh trong Python nằm trên một dòng mã nguồn. Ta không cần phải kết thúc câu lệnh bằng bất kì kí tự gì. Cũng như các ngôn ngữ khác, Python cũng có các cấu trúc điều khiển. Chúng bao gồm:

Cấu trúc rẽ nhánh: cấu trúc if (có thể sử dụng thêm elif hoặc else), dùng để thực thi có điều kiện một khối mã cụ thể.

Cấu trúc lặp, bao gồm:

- Lệnh while: chạy một khối mã cụ thể cho đến khi điều kiện lặp có giá trị false.
- Vòng lặp for: lặp qua từng phần tử của một dãy, mỗi phần tử sẽ được đưa vào biến cục bộ để sử dụng với khối mã trong vòng lặp.

Python cũng có từ khóa class dùng để khai báo lớp (sử dụng trong lập trình hướng đối tượng) và lệnh def dùng để định nghĩa hàm.

- Hệ thống kiểu dữ liệu

Python sử dụng hệ thống kiểu duck typing, còn gọi là latent typing (tự động xác định kiểu). Có nghĩa là, Python không kiểm tra các ràng buộc về kiểu dữ liệu tại thời điểm dịch, mà là tại thời điểm thực thi. Khi thực thi, nếu một thao tác trên một đối tượng bị thất bại, thì có nghĩa là đối tượng đó không sử dụng một kiểu thích hợp.

Python cũng là một ngôn ngữ định kiểu mạnh. Nó cấm mọi thao tác không hợp lệ, ví dụ cộng một con số vào chuỗi kí tự.

Sử dụng Python, ta không cần phải khai báo biến. Biến được xem là đã khai báo nếu nó được gán một giá trị lần đầu tiên. Căn cứ vào mỗi lần gán, Python sẽ tự động xác định kiểu dữ liệu của biến. Python có một số kiểu dữ liệu thông dụng sau:

- int, long: số nguyên (trong phiên bản 3.x long được nhập vào trong kiểu int). Độ dài của kiểu số nguyên là tùy ý, chỉ bị giới hạn bởi bộ nhớ máy tính.
- float: số thực
- complex: số phức, chẳng hạn $5+4j$

- list: dãy trong đó các phần tử của nó có thể được thay đổi, chẳng hạn [8, 2, 'b', -1.5]. Kiểu dãy khác với kiểu mảng (array) thường gặp trong các ngôn ngữ lập trình ở chỗ các phần tử của dãy không nhất thiết có kiểu giống nhau. Ngoài ra phần tử của dãy còn có thể là một dãy khác.
- tuple: dãy trong đó các phần tử của nó không thể thay đổi.
- str: chuỗi kí tự. Từng kí tự trong chuỗi không thể thay đổi. Chuỗi kí tự được đặt trong dấu nháy đơn, hoặc nháy kép.
- dict: từ điển, còn gọi là "hashtable": là một cặp các dữ liệu được gắn theo kiểu {từ khóa: giá trị}, trong đó các từ khóa trong một từ điển nhất thiết phải khác nhau. Chẳng hạn {1: "Python", 2: "Pascal"}
- set: một tập không xếp theo thứ tự, ở đó, mỗi phần tử chỉ xuất hiện một lần.

Ngoài ra, Python còn có nhiều kiểu dữ liệu khác. Xem thêm trong phần "Các kiểu dữ liệu" bên dưới.

- Module

Python cho phép chia chương trình thành các module để có thể sử dụng lại trong các chương trình khác. Nó cũng cung cấp sẵn một tập hợp các modules chuẩn mà lập trình viên có thể sử dụng lại trong chương trình của họ. Các module này cung cấp nhiều chức năng hữu ích, như các hàm truy xuất tập tin, các lời gọi hệ thống, trợ giúp lập trình mạng (socket),...

- Đa năng

Python là một ngôn ngữ lập trình đơn giản nhưng rất hiệu quả.

- So với Unix shell, Python hỗ trợ các chương trình lớn hơn và cung cấp nhiều cấu trúc hơn.
- So với C, Python cung cấp nhiều cơ chế kiểm tra lỗi hơn. Nó cũng có sẵn nhiều kiểu dữ liệu cấp cao, ví dụ như các mảng (array) linh hoạt và từ điển (dictionary) mà ta sẽ phải mất nhiều thời gian nếu viết bằng C.

Python là một ngôn ngữ lập trình cấp cao có thể đáp ứng phần lớn yêu cầu của lập trình viên:

- Python thích hợp với các chương trình lớn hơn cả AWK và Perl.
- Python được sử dụng để lập trình Web. Nó có thể được sử dụng như một ngôn ngữ kịch bản.
- Python được thiết kế để có thể nhúng và phục vụ như một ngôn ngữ kịch bản để tuỳ biến và mở rộng các ứng dụng lớn hơn.
- Python được tích hợp sẵn nhiều công cụ và có một thư viện chuẩn phong phú, Python cho phép người dùng dễ dàng tạo ra các dịch vụ Web, sử dụng các thành phần COM hay CORBA, hỗ trợ các loại định dạng dữ liệu Internet như email, HTML, XML và các ngôn ngữ

đánh dấu khác. Python cũng được cung cấp các thư viện xử lý các giao thức Internet thông dụng như HTTP, FTP,...

- Python có khả năng giao tiếp đến hầu hết các loại cơ sở dữ liệu, có khả năng xử lý văn bản, tài liệu hiệu quả, và có thể làm việc tốt với các công nghệ Web khác.
- Python đặc biệt hiệu quả trong lập trình tính toán khoa học nhờ các công cụ Python Imaging Library, pyVTK, MayaVi 3D Visualization Toolkits, Numeric Python, ScientificPython,...
- Python có thể được sử dụng để phát triển các ứng dụng desktop. Lập trình viên có thể dùng wxPython, PyQt, PyGtk để phát triển các ứng dụng giao diện đồ họa (GUI) chất lượng cao. Python còn hỗ trợ các nền tảng phát triển phần mềm khác như MFC, Carbon, Delphi, X11, Motif, Tk, Fox, FLTK, ...
- Python cũng có sẵn một unit testing framework để tạo ra các bộ test (test suites).
 - Multiple paradigms (đa biến hóa)

Python là một ngôn ngữ đa biến hóa (multiple paradigms). Có nghĩa là, thay vì ép buộc mọi người phải sử dụng duy nhất một phương pháp lập trình, Python lại cho phép sử dụng nhiều phương pháp lập trình khác nhau: hướng đối tượng, có cấu trúc, chức năng, hoặc chỉ hướng đến một khía cạnh. Python kiểu động và sử dụng bộ thu gom rác để quản lý bộ nhớ. Một đặc điểm quan trọng nữa của Python là giải pháp tên động, kết nối tên biến và tên phương thức lại với nhau trong suốt thực thi của chương trình.

- Sự tương đương giữa true và một giá trị khác 0

Cũng như C/C++, bất kì một giá trị khác 0 nào cũng tương đương với true và ngược lại, một giá trị 0 tương đương với false. Như vậy:

```
if a != 0:
```

tương đương với:

```
if a:
```

- Cú pháp

Sau đây là cú pháp cơ bản nhất của ngôn ngữ Python:

- + *Toán tử*

```
+ - * / // (chia làm tròn) % (phần dư) ** (lũy thừa)
~ (not) & (and) | (or) ^ (xor)
<< (left shift) >> (right shift)
== (bằng) <= >= != (khác)
```

Python sử dụng kí pháp trung tố thường gặp trong các ngôn ngữ lập trình khác.

+ Các kiểu dữ liệu

 ✖ Kiểu số

```
1234585396326 (số nguyên dài vô hạn) -86.12 7.84E-04  
2j 3 + 8j (số phức)
```

 ✖ Kiểu chuỗi (string)

```
"Hello" "It's me" '"OK"-he replied'
```

 ✖ Kiểu bộ (tuple)

```
(1, 2.0, 3) (1,) ("Hello",1,())
```

 ✖ Kiểu danh sách (list)

```
[4.8, -6] ['a', 'b']
```

 ✖ Kiểu từ điển (dictionary)

```
{"Vietnam": "Hanoi", "Netherlands": "Amsterdam", "France": "Paris"}
```

+ Chú thích

```
# dòng chú thích
```

+ Lệnh gán

```
tên biến = biểu thức  
x = 23.8  
y = -x ** 2  
z1 = z2 = x + y  
loiChao = "Hello!"  
  
i += 1 # tăng biến i thêm 1 đơn vị
```

+ In giá trị

```
print biểu thức  
print (7 + 8) / 2.0  
print (2 + 3j) * (4 - 6j)
```

+ Nội suy chuỗi (string interpolation)

```
print "Hello %s" %("world!")  
print "i = %d" %i
```

```
print "a = %.2f and b = %.3f" %(a,b)
```

+ *Cấu trúc rẽ nhánh*

☞ *Dạng 1:*

```
if biểu_thức_điều_kiện:  
    # lệnh ...
```

☞ *Dạng 2:*

```
if biểu_thức_điều_kiện:  
    # lệnh ...  
else:  
    # lệnh ...
```

☞ *Dạng 3:*

```
if biểu_thức_điều_kiện_1:  
    # lệnh ... (được thực hiện nếu biểu_thức_điều_kiện_1 là đúng/true)  
elif biểu_thức_điều_kiện_2:  
    # lệnh ... (được thực hiện nếu biểu_thức_điều_kiện_1 là sai/false,  
nhưng biểu_thức_điều_kiện_2 là đúng/true)  
else:  
    # lệnh ... (được thực hiện nếu tất cả các biểu thức điều kiện đi kèm if  
và elif đều sai)
```

+ *Cấu trúc lặp*

```
while biểu_thức_đúng:  
    # lệnh ...  
    for phần_tử_in_dãy:  
        # lệnh ...  
        L = ["Ha Noi", "Hai Phong", "TP Ho Chi Minh"]  
        for thanhPho in L:  
            print thanhPho  
  
for i in range(10):  
    print i
```

+ *Hàm*

```
def tên_hàm (tham_biến_1, tham_biến_2, tham_biến_n):  
    # lệnh ...  
    return giá_trí_hàm  
def bìnhPhuong(x):  
    return x*x
```

+ *Hàm với tham số mặc định:*

```
def luyThua(x, n=2):  
    """Lũy thừa với số mũ mặc định là 2"""  
    return x**n
```

```
print luyThua(3) # 9
    print luyThua(2,3) # 8
```

+ Lớp

```
class Tên_Lớp_1:
    # ...

class Tên_Lớp_2(Tên_Lớp_1):
    """Lớp 2 kế thừa lớp 1"""
    x = 3 # biến thành viên của lớp
    #
    def phương_thức(self, tham_biến):
        # ...

# khởi tạo
a = Tên_Lớp_2()
print a.x
print a.phương_thức(m) # m là giá trị gán cho tham biến
List Comprehension
```

+ List Comprehension

Là dạng cú pháp đặc biệt (syntactic sugar) (mới có từ Python 2.x) cho phép thao tác trên toàn bộ dãy (list) mà không cần viết rõ vòng lặp. Chẳng hạn y là một dãy mà mỗi phần tử của nó bằng bình phương của từng phần tử trong dãy x:

```
y = [xi**2 for xi in x]
```

+ Xử lý lỗi

```
try:
    câu_lệnh
except Loại_Lỗi:
    thông báo lỗi
```

+ Tốc độ thực hiện

Là một ngôn ngữ thông dịch, Python có tốc độ thực hiện chậm hơn nhiều lần so với các ngôn ngữ biên dịch như Fortran, C, v.v... Trong số các ngôn ngữ thông dịch, Python được đánh giá nhanh hơn Ruby và Tcl, nhưng chậm hơn Lua.

+ Các đặc điểm mới trong Python 3.x

Nội dung phần này được trích từ tài liệu của Guido van Rossum. Phần này không liệt kê đầy đủ tất cả các đặc điểm; chi tiết xin xem tài liệu nói trên.

+ Một số thay đổi cần lưu ý nhất

Lệnh print trở thành hàm print. Theo đó sau print() ta cần nhớ gõ vào cặp ngoặc ():

```
print("Hello")
print(2+3)
```

Trả lại kết quả không còn là list trong một số trường hợp.

- dict.keys(), dict.items(), dict.values() kết quả cho ra các "view" thay vì list.
- map và filter trả lại các iterator.
- range bây giờ có tác dụng như xrange, và không trả lại list.

☞ So sánh

Không còn hàm cmp, và cmp(a, b) có thể được thay bằng ($a > b$) - ($a < b$)

☞ Số nguyên

- Kiểu long được đổi tên thành int.
- 1/2 cho ta kết quả là số thực chứ không phải số nguyên.
- Không còn hằng số sys.maxint
- Kiểu bát phân được kí hiệu bằng 0o thay vì 0, chẳng hạn 0o26.

Phân biệt văn bản - dữ liệu nhị phân thay vì Unicode - chuỗi 8-bit

- Tất cả chuỗi văn bản đều dưới dạng Unicode, nhưng chuỗi Unicode mã hóa lại là dạng dữ liệu nhị phân. Dạng mặc định là UTF-8.
- Không thể viết u"a string" để biểu diễn chuỗi như trong các phiên bản 2.x

+ Các thay đổi về cú pháp

☞ Cú pháp mới

- Các tham biến chỉ chấp nhận keyword: Các tham biến phía sau *args phải được gọi theo dạng keyword.
- Từ khóa mới nonlocal. Muốn khai báo một biến x với phạm vi ảnh hưởng rộng hơn, nhưng chưa đến mức toàn cục, ta dùng nonlocal x.
- Gán giá trị vào các phần tử tuple một cách thông minh, chẳng hạn có thể viết (a, *rest, b) = range(5) để có được a = 0; b = [1,2,3]; c = 4.
- Dictionary comprehension, chẳng hạn {k: v for k, v in stuff} thay vì dict(stuff).
- Kiểu nhị phân, chẳng hạn b110001.

❖ Cú pháp được thay đổi

- raise [biểu_thức [from biểu_thức]]
- except lệnh as biến
- Sử dụng metaclass trong đối tượng:

```
class C(metaclass=M):  
    pass
```

Cách dùng biến `__metaclass__` không còn được hỗ trợ.

+ Cú pháp bị loại bỏ

- Không còn dấu ``, thay vì đó, dùng repr.
- Không còn so sánh <> (dùng !=).

Không còn các lớp kiểu classic.

1.2. Hướng dẫn cài đặt Ananconda

1.2.1. Giới thiệu Ananconda

Anaconda là nền tảng mã nguồn mở về Khoa học dữ liệu trên Python thông dụng nhất hiện nay. Anaconda hướng đến việc quản lý các package một cách đơn giản, phù hợp với mọi người. Hệ thống quản lý package của Anaconda là Conda. Anaconda Với hơn 11 triệu người dùng, Anaconda là cách nhanh nhất và dễ nhất để học Khoa học dữ liệu với Python hoặc R trên Windows, Linux và Mac OS X. Lợi ích của Anaconda:

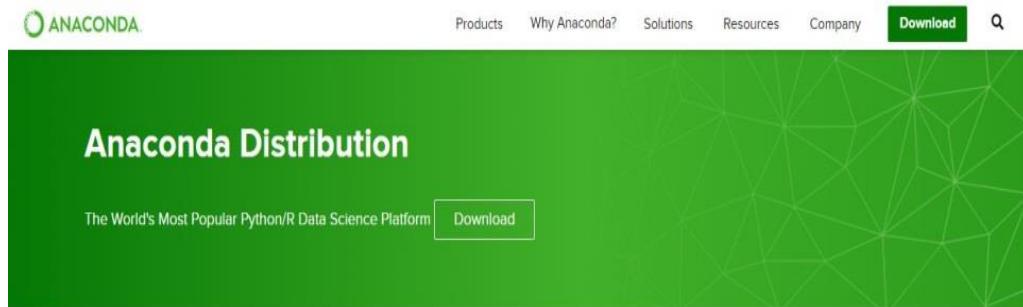
- Dễ dàng tải 1500+ packages về Python/R cho data science
- Quản lý thư viện, môi trường và dependency giữa các thư viện dễ dàng
- Dễ dàng phát triển mô hình machine learning và deep learning với scikit-learn, tensorflow, keras
- Xử lý dữ liệu tốc độ cao với numpy, pandas
- Hiện thị kết quả với Matplotlib, Bokeh

Trong khi đó Spyder là 1 trong những IDE (môi trường tích hợp dùng để phát triển phần mềm) tốt nhất cho data science và quang trọng hơn là nó được cài đặt khi bạn cài đặt Anaconda.

Yêu cầu phần cứng và phần mềm:

- Hệ điều hành: Win 7, Win 8/8.1, Win 10, Red Hat Enterprise Linux/CentOS 6.7, 7.3, 7.4, and 7.5, and Ubuntu 12.04+.
- Ram tối thiểu 4GB

Ổ cứng trống tối thiểu 3GB để tải và cài đặt



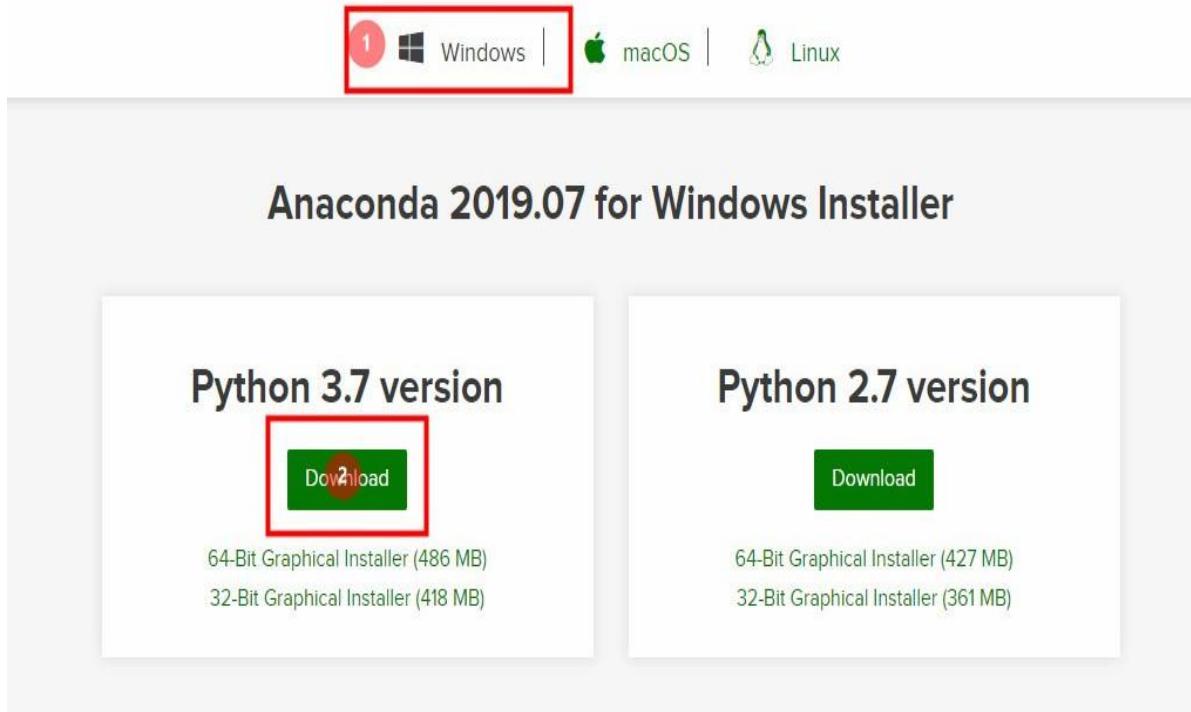
The open-source Anaconda Distribution is the easiest way to perform Python/R data science and machine learning on Linux, Windows, and Mac OS X. With over 15 million users worldwide, it is the industry standard for developing, testing, and training on a single machine, enabling *individual data scientists* to:

- Quickly download 1,500+ Python/R data science packages
- Manage libraries, dependencies, and environments with Conda
- Develop and train machine learning and deep learning models with scikit-learn, TensorFlow, and Theano
- Analyze data with scalability and performance with Dask, NumPy, pandas, and Numba
- Visualize results with Matplotlib, Bokeh, Databricks, and Holoviews

Hình 1.2: Giao diện trang chủ Anaconda

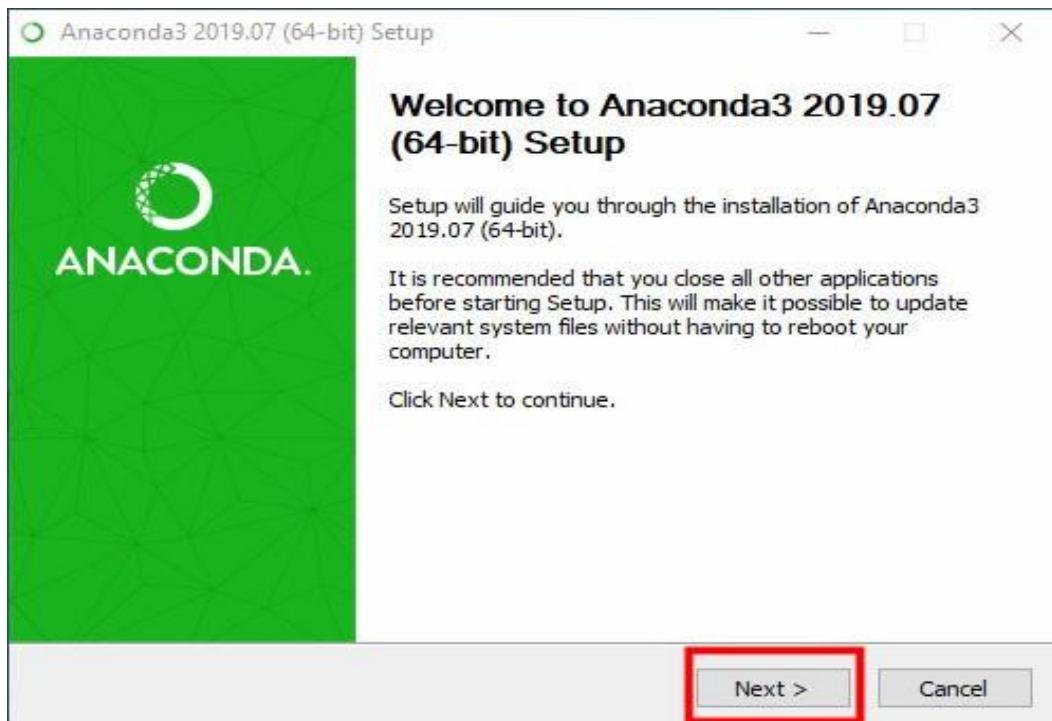
1.2.2. Download Anaconda và hướng dẫn cài đặt trên HĐH Window

- ☞ Download Anaconda (python3) cho HĐH Window vè tại <https://www.anaconda.com/distribution/>
- ☞ Hiện tại phiên bản Python mà ta sử dụng là python3.7 bản phân phối Anaconda 2019.07

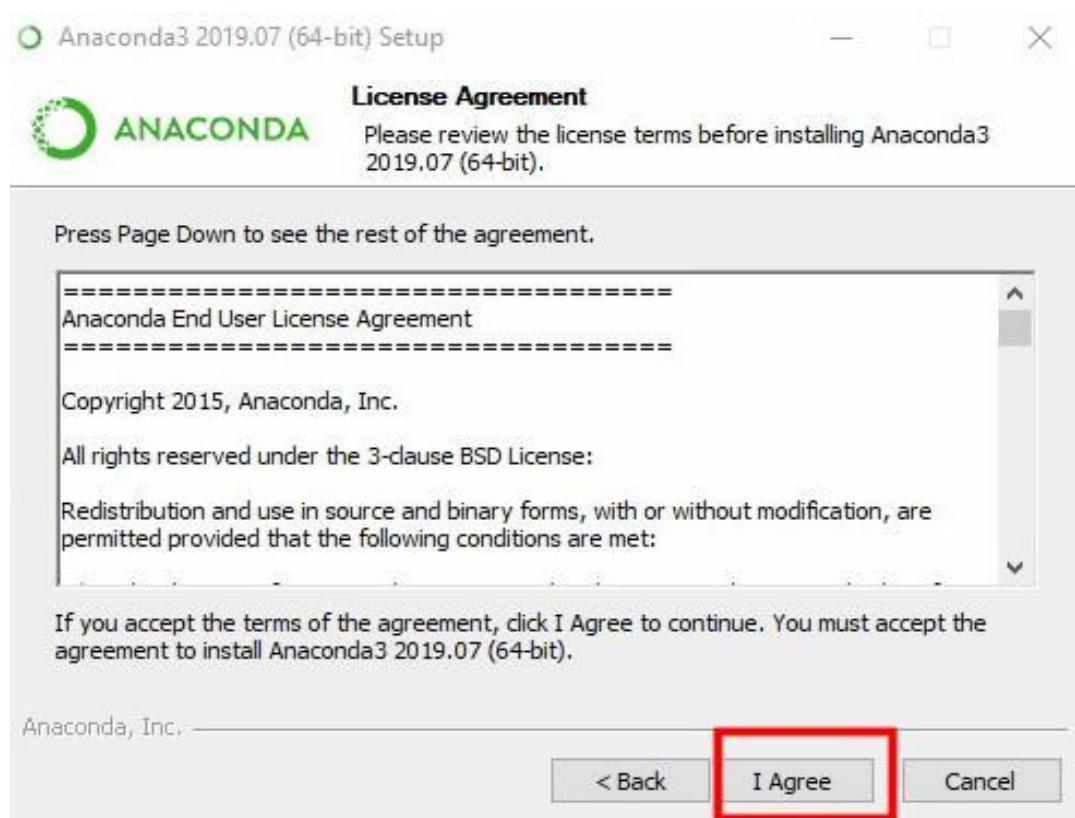


Hình 1.3: Trang tải Anaconda

Sau khi tải về xong bạn chạy file “Anaconda3-2019.07-Windows-x86_64.exe”

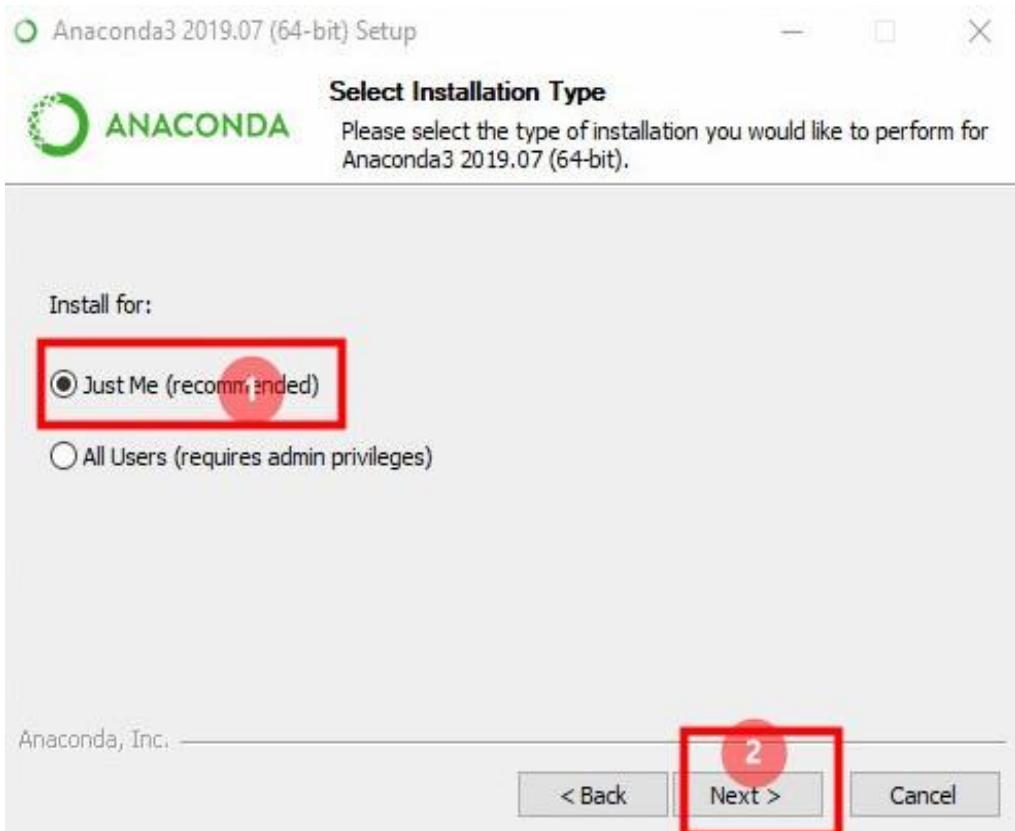


Hình 1. 4: Chạy chương trình vừa tải về



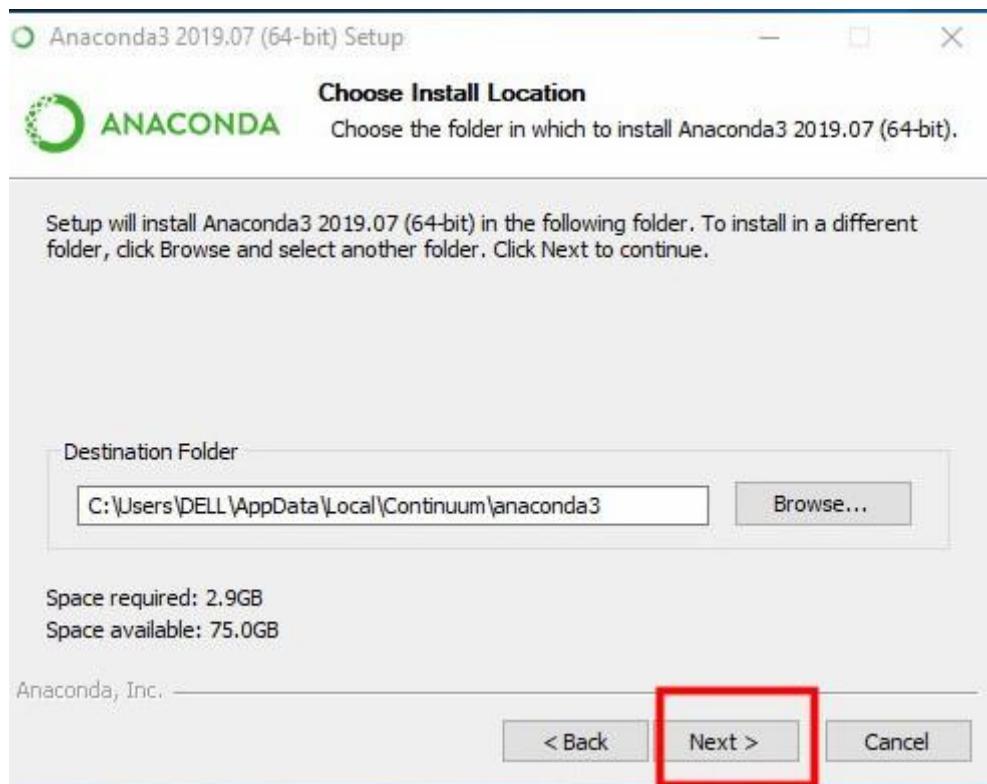
Hình 1. 5: License Agreement

- Click I Agree



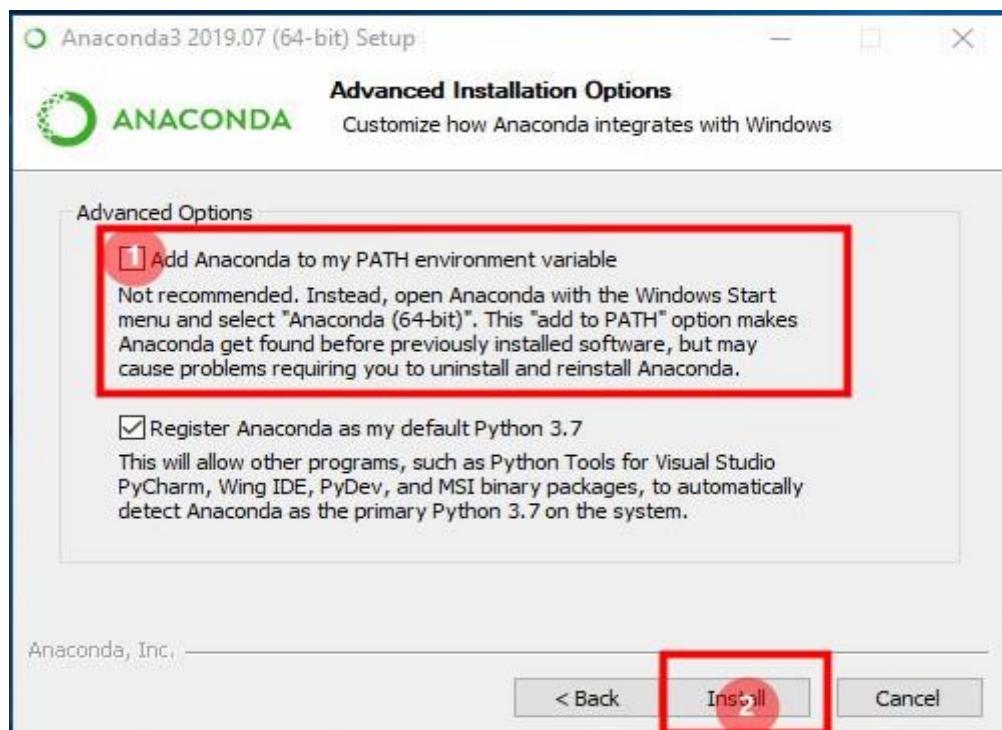
Hình 1.6: Chọn Loại cài đặt

- Bạn chọn “Just Me” sau đó Click Next



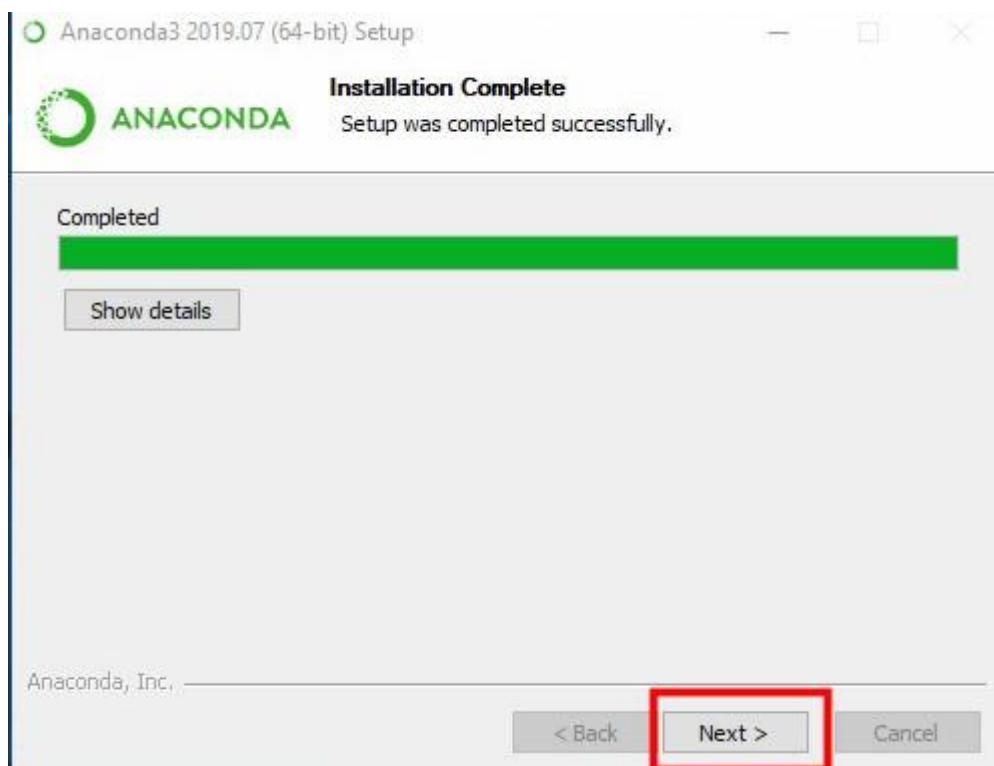
Hình 1. 7: Chọn Vị trí cài đặt

- Click Next



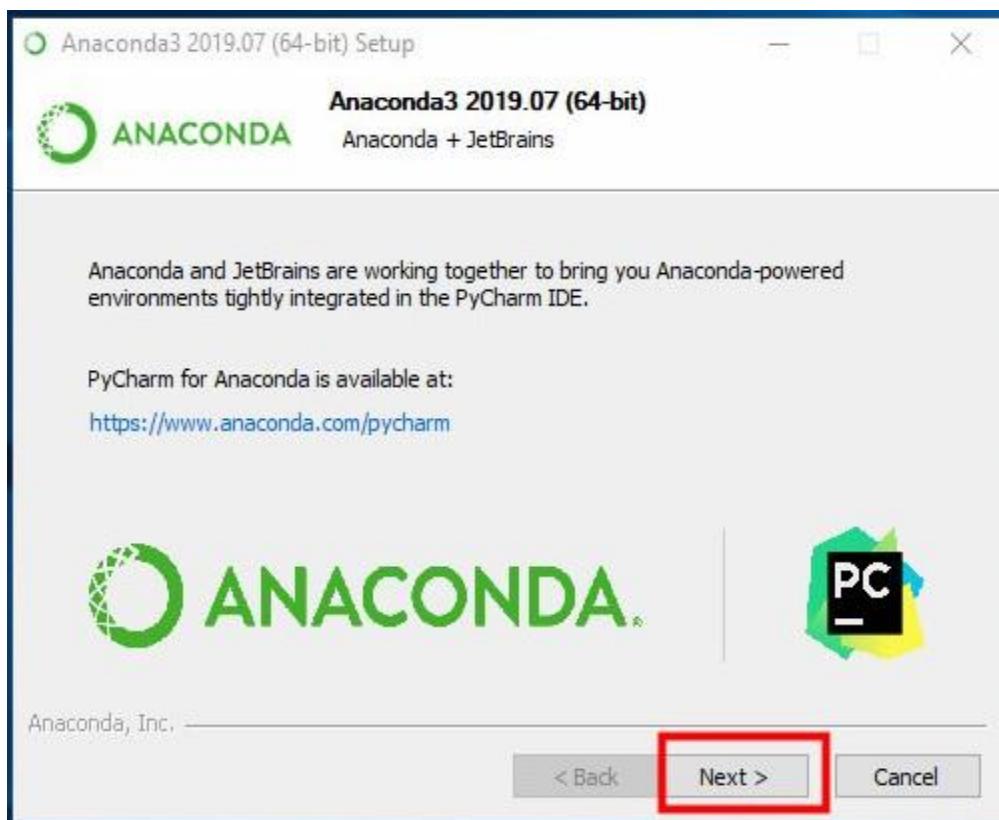
Hình 1. 8: Advanced Options

- Lưu ý: hãy chắc là bạn chọn tùy chọn được khoanh đỏ bên dưới để thêm các câu lệnh Anaconda vào **System Environment Variable (PATH)** của Windows
- Sau đó Click **Next**



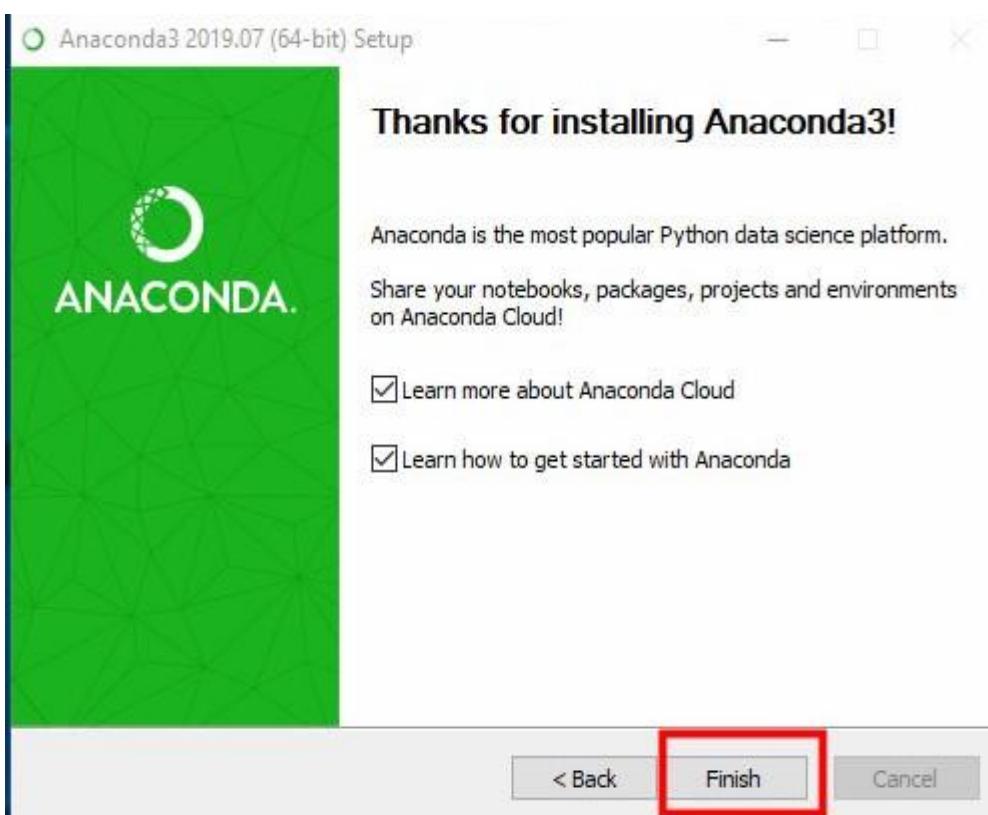
Hình 1. 9: Hoàn thành Cài đặt

- Tiếp tục Click **Next**



Hình 1. 10: Anaconda và JetBrains

- Click Next



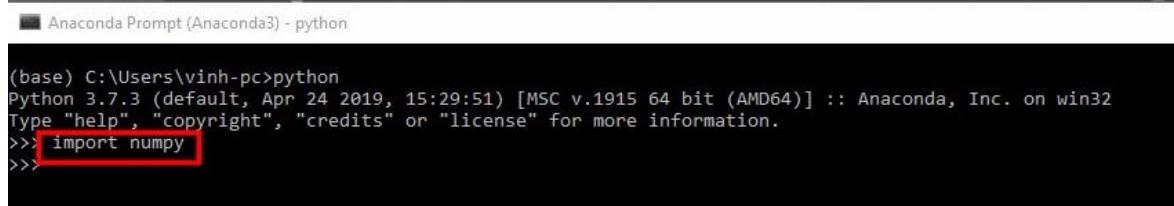
Hình 1.11: Cài đặt hoàn tất và khởi động chương trình

- Tiếp theo bạn click **Finish** để hoàn tất việc cài đặt

1.2.3. Hướng dẫn cài đặt thêm thư viện

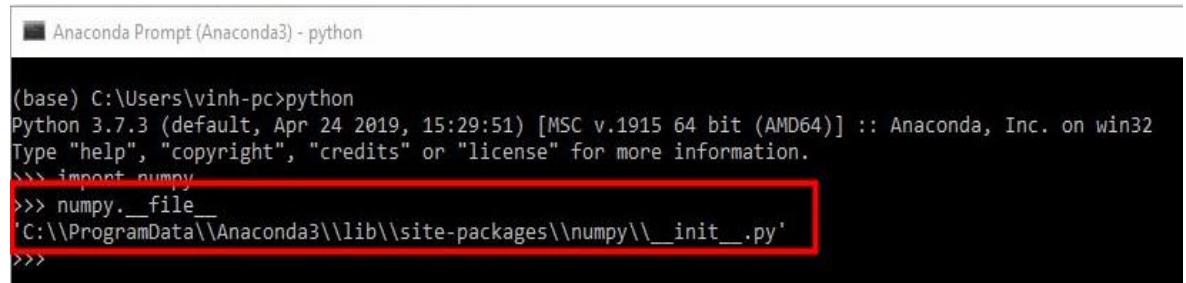
Anaconda đã có sẵn khá là nhiều thư viện python như : [Numpy](#), [Scipy](#), [Matplotlib](#), [sklearn](#),...

Để kiểm tra python của Anaconda đã có thư viện nào đó, chúng ta sẽ thử import nó trong Console



```
Anaconda Prompt (Anaconda3) - python
(base) C:\Users\vinh-pc>python
Python 3.7.3 (default, Apr 24 2019, 15:29:51) [MSC v.1915 64 bit (AMD64)] :: Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import numpy
>>>
```

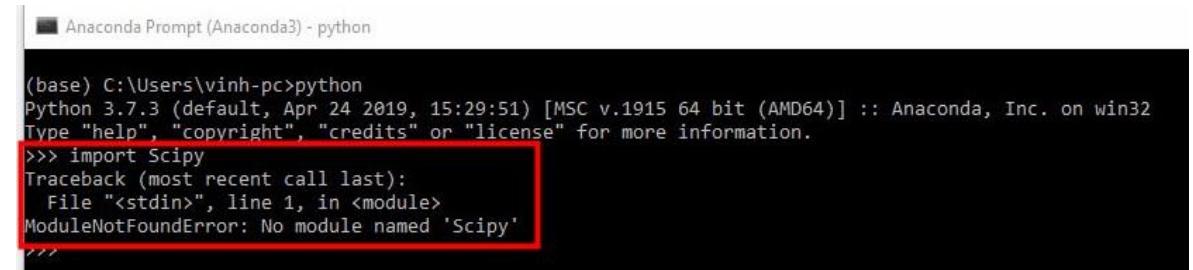
Không có lỗi được thông báo nghĩa là python đã biết được thư viện này. Để kiểm tra thư viện này ở đâu, sau khi *import*, ta truy xuất đường dẫn của thư viện như sau:



```
Anaconda Prompt (Anaconda3) - python
(base) C:\Users\vinh-pc>python
Python 3.7.3 (default, Apr 24 2019, 15:29:51) [MSC v.1915 64 bit (AMD64)] :: Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import numpy
>>> numpy.__file__
'C:\ProgramData\Anaconda3\lib\site-packages\numpy\_\_init__.py'
>>>
```

Thư viện Numpy của tôi nằm ở đường dẫn '<C:\ProgramData\Anaconda3\lib\site-packages>'.

Anaconda đã có sẵn thư viện Numpy



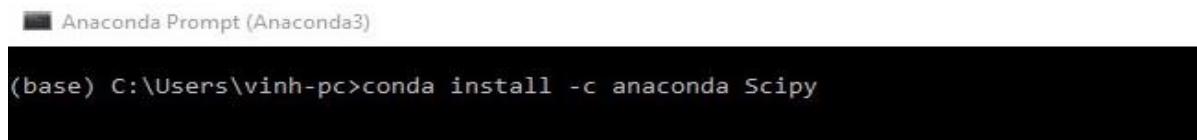
```
Anaconda Prompt (Anaconda3) - python
(base) C:\Users\vinh-pc>python
Python 3.7.3 (default, Apr 24 2019, 15:29:51) [MSC v.1915 64 bit (AMD64)] :: Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import Scipy
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
ModuleNotFoundError: No module named 'Scipy'
>>>
```

Nếu như Python trả về lỗi Import như trên thì có nghĩa trong Anaconda chúng ta chưa có thư viện đó.

Ở phần trên python của tôi chưa có thư viện *Scipy*, nên tôi phải đi cài đặt nó. Vì tôi sử dụng Anaconda cho lập trình python nên tôi cần phải (1) *cài đặt thư viện mới vào đường dẫn libs python của Anaconda* hoặc (2) *chỉ cho python của Anaconda biết về đường dẫn tới thư viện mới này*.

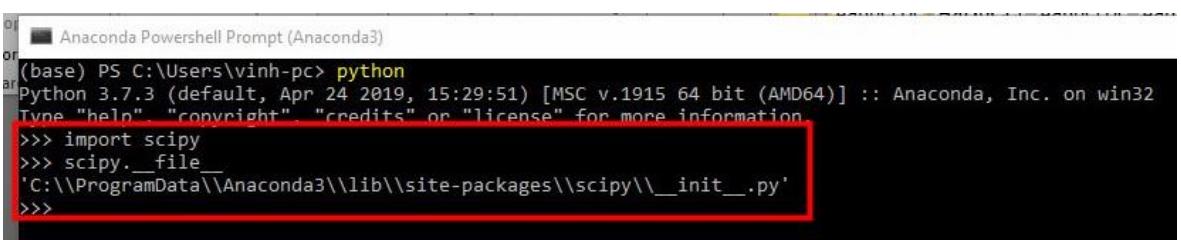
Với Anaconda, việc cài đặt 1 thư viện đang được hỗ trợ cực kỳ đơn giản, tôi chỉ cần dùng tools *pip* hoặc *conda* mà Anaconda đã cài sẵn. Cụ thể, ở đây tôi muốn cài thư viện *Scipy* tôi truy cập vào trang chủ của [Scipy](#). Trang này ghi rằng chúng ta có thể cài bằng *pip* hoặc *conda*.

Chúng ta sẽ bật Anaconda Prompt (Anaconda3) lên và gõ `conda install -c anaconda Scipy`. Conda sẽ tự động tìm thư viện *Scipy* và cài vào đường dẫn Anaconda giúp chúng ta.



```
Anaconda Prompt (Anaconda3)
(base) C:\Users\vinh-pc>conda install -c anaconda Scipy
```

Chờ cho thư viện và các thư viện liên quan hoàn tất cài đặt, chúng ta vào spyder kiểm tra lại đã có *Scipy* chưa. Và python trả về đã có *Scipy* trong Anaconda. Và chúng ta đã có thể sử dụng *Scipy*



```
Anaconda Powershell Prompt (Anaconda3)
(base) PS C:\Users\vinh-pc> python
Python 3.7.3 (default, Apr 24 2019, 15:29:51) [MSC v.1915 64 bit (AMD64)] :: Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license" for more information
>>> import scipy
>>> scipy.__file__
'C:\ProgramData\Anaconda3\lib\site-packages\scipy\__init__.py'
>>>
```

Với 1 thư viện chưa có trên Anaconda, cách cài đặt sẽ phức tạp hơn chút nhưng hầu hết các thư viện lớn thường dùng đều có thể cài đặt thông qua Anaconda, nên chúng ta không phải lo lắng lắm.

Ngoài ra chúng ta có thể cài đặt thêm các thư viện bằng Anaconda Navigator

1.3. Hướng dẫn cài đặt IDE Spyder

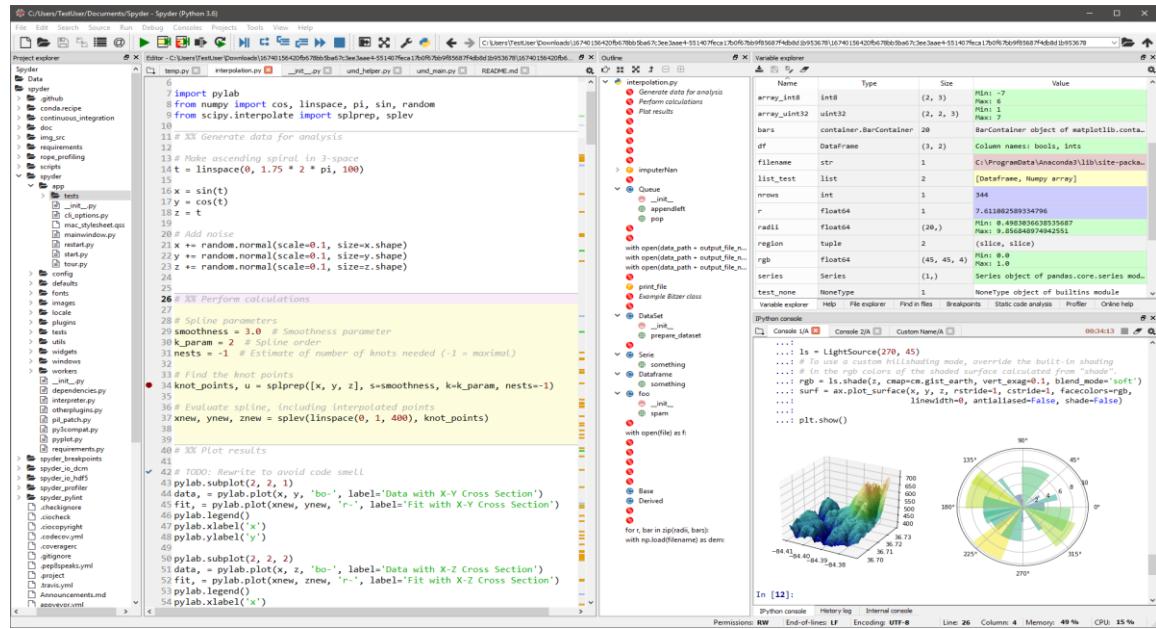
1.3.1. Giới thiệu IDE Spyder

Spyder là một môi trường phát triển Python mã nguồn mở được tối ưu hóa cho các bài toán liên quan đến khoa học dữ liệu. Spyder đi kèm với phân phối quản lý gói Anaconda. Spyder là công cụ thường dùng của các nhà khoa học dữ liệu sử dụng Python. Spyder tích hợp tốt với các thư viện khoa học dữ liệu Python phổ biến như SciPy, NumPy và Matplotlib.

Spyder có hầu hết các tính năng của một “IDE phổ biến”, chẳng hạn như trình soạn thảo mã với chức năng đánh dấu cú pháp mạnh mẽ, tự động hoàn thành mã và thậm chí là trình duyệt tài liệu được tích hợp.

Một tính năng đặc biệt không có trong các môi trường phát triển Python khác là tính năng “khám phá biến” của Spyder cho phép hiển thị dữ liệu bằng cách sử dụng bố cục bảng ngay bên trong IDE. Điều này làm nó nhanh chóng và gọn gàng. Nếu bạn thường xuyên làm các bài toán khoa học dữ liệu làm việc bằng cách sử dụng Python, thì đây là một tính năng độc đáo. Việc tích hợp IPython/Jupyter là một đặc điểm nổi bật khác.

Nhìn chung Spyder có nhiều chức năng nổi trội cơ bản hơn các IDE khác. Điều đặc biệt khác là Spyder miễn phí trên Windows, macOS, và Linux và nó là phần mềm mã nguồn mở hoàn toàn.



Hình 1. 12: Giao diện chính của IDE Spyder

- ✓ **Ưu điểm:** Tối ưu nhiều tính chất phù hợp cho các hoạt động khoa học dữ liệu sử dụng phân phối Python Anaconda.
- ✓ **Nhược điểm:** Các nhà phát triển Python có kinh nghiệm hơn có thể cảm thấy Spyder quá đơn giản để làm việc hàng ngày và thay vào đó chọn một IDE hoàn chỉnh hơn hoặc một giải pháp biên tập có khả năng tùy chỉnh.

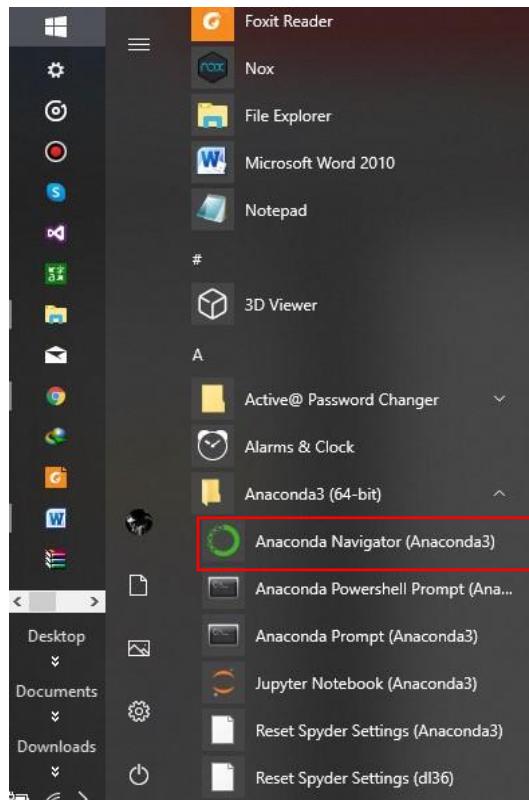
1.3.2. Hướng dẫn cài đặt IDE Spyder bằng Navigator

Spyder tương đối dễ cài đặt trên Windows, Linux và macOS. Chỉ cần chắc chắn để đọc và làm theo các hướng dẫn cẩn thận.

Spyder được bao gồm theo mặc định trong bản phân phối Anaconda Python, đi kèm với mọi thứ bạn cần để bắt đầu trong gói tất cả trong một (thường là khi cài đặt Anaconda thì Spyder đã được mặc định cài đặt).

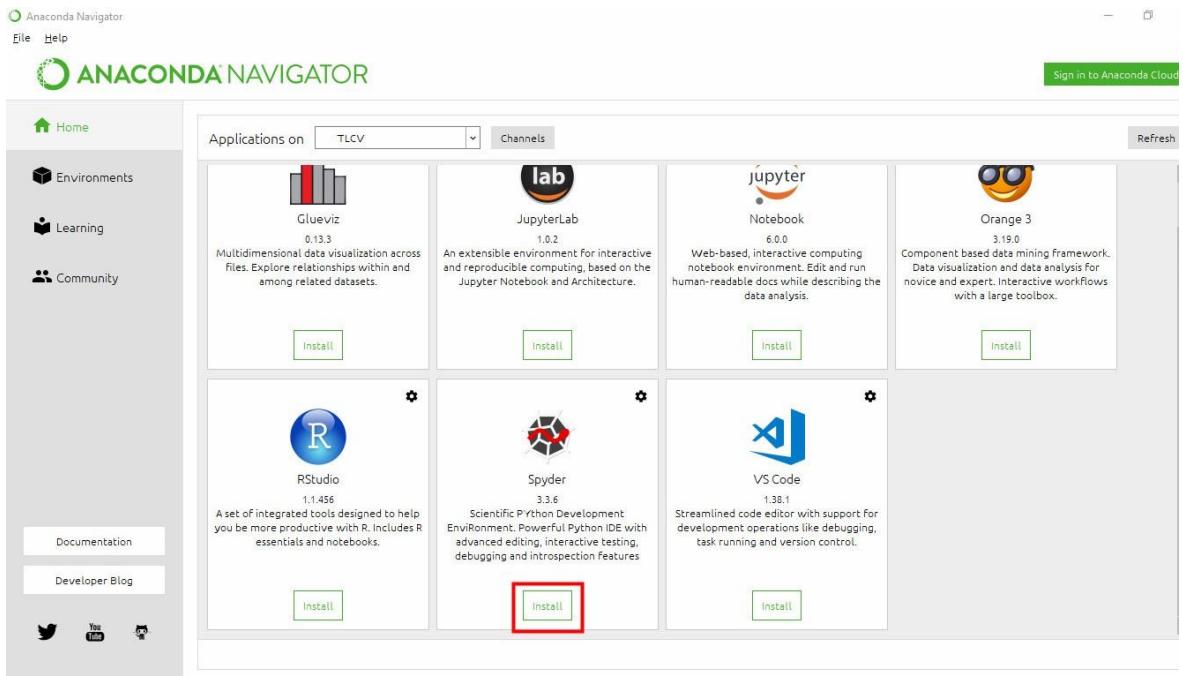
Đây là cách dễ nhất để cài đặt Spyder cho bất kỳ nền tảng được hỗ trợ nào và là cách khuyên bạn nên tránh các sự cố không mong muốn. Bạn nên cài đặt thông qua phương pháp này; nó thường có ít khả năng gây ra những cạm bẫy tiềm tàng cho những người không phải là chuyên gia và có thể cung cấp hỗ trợ hạn chế nếu gặp rắc rối.

Đầu tiên bạn cần khởi động Anaconda Navigator



Hình 1. 13:Chạy Spyder

Sau đó bạn vào phần Home rồi cài đặt Spyder theo phiên bản mà Anaconda hỗ trợ

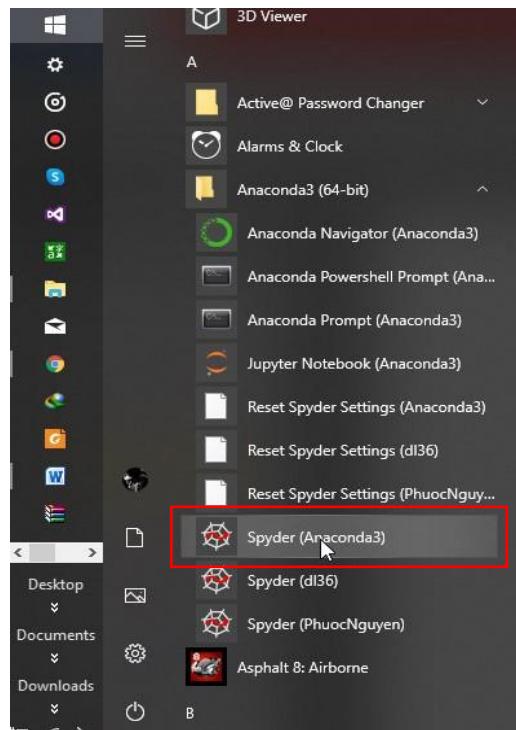


Hình 1. 14:Cài đặt Spyder bằng Anaconda Navigator

Chờ ít phút để chương trình cài đặt hoàn thành.

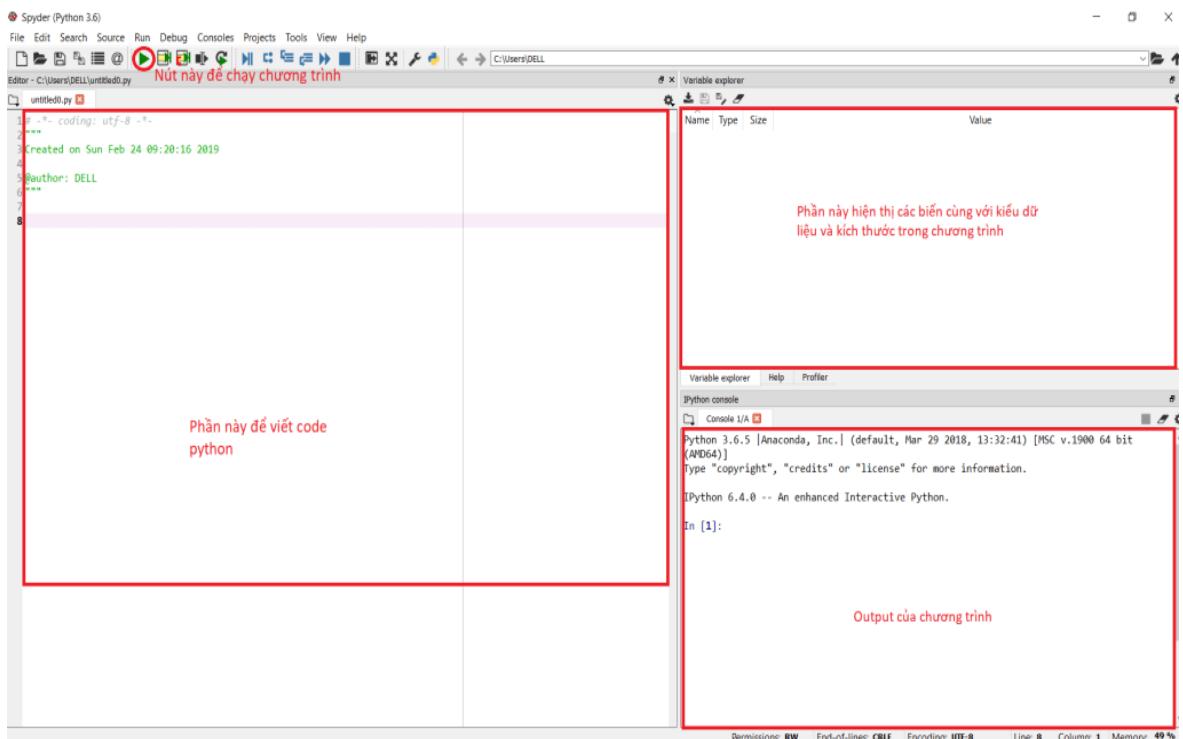
1.3.4. Hướng dẫn sử dụng Spyder

Sau khi cài đặt Spyder hoàn thành, mở Spyder lên và sử dụng.



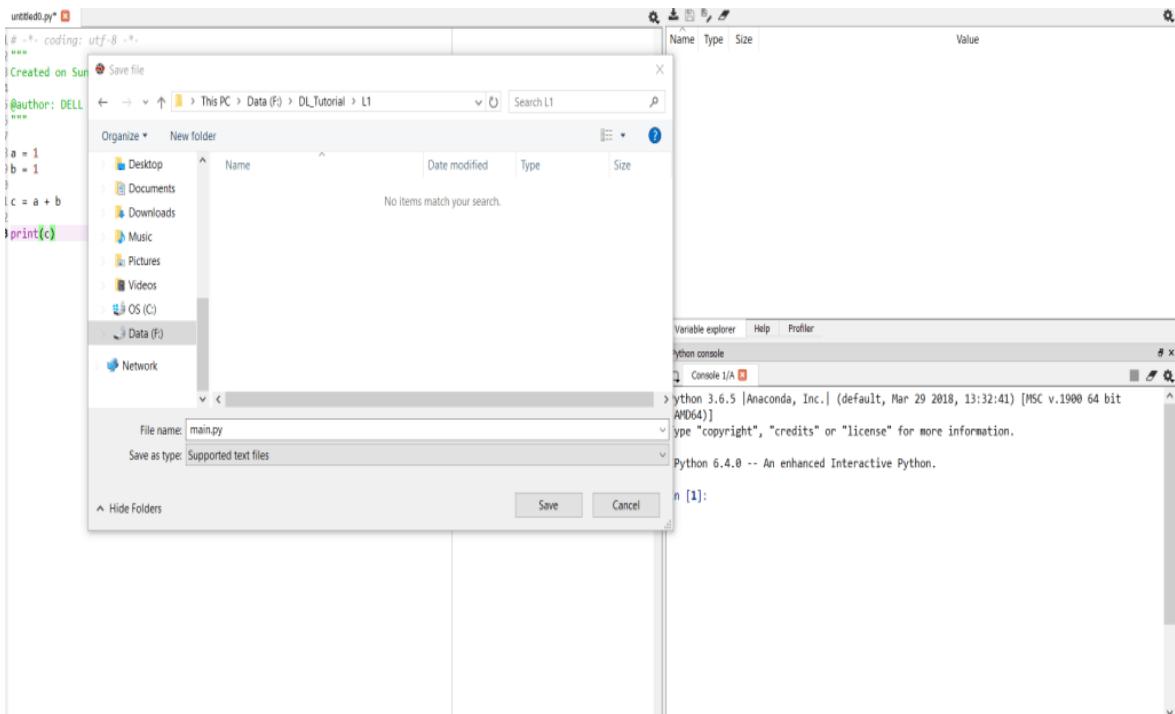
Hình 1. 15: Mở chương trình Spyder

Giao diện chính của Spyder



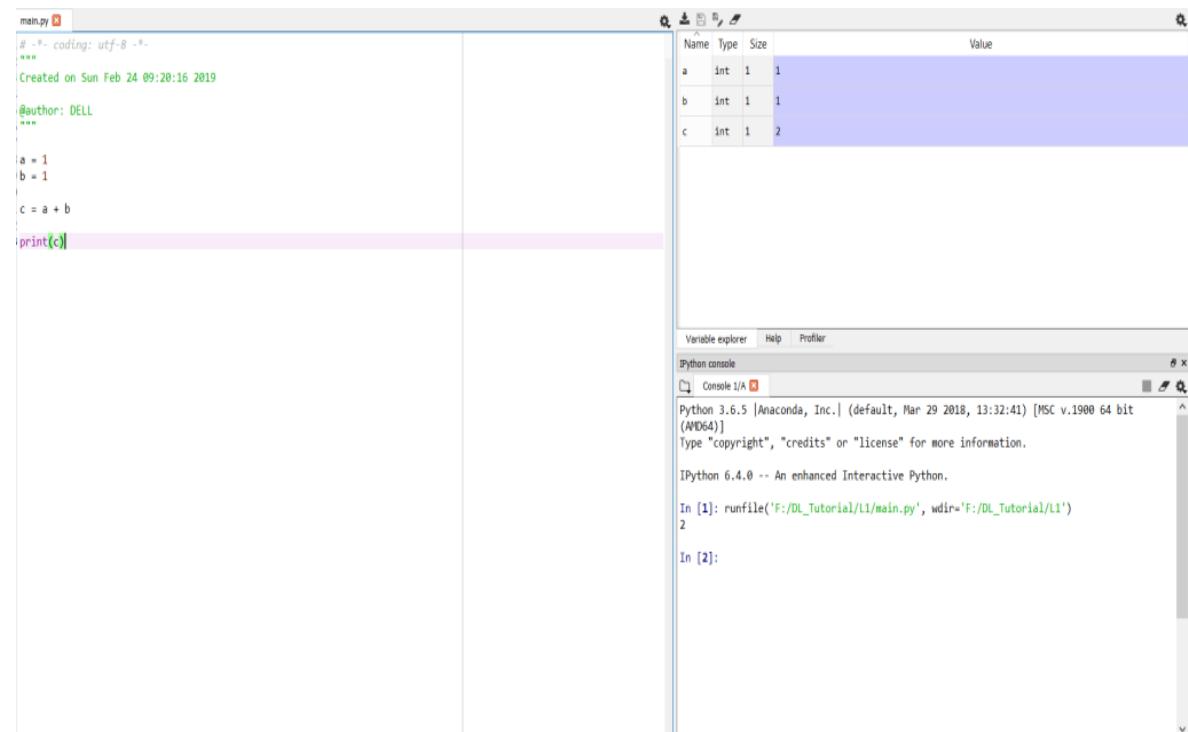
Hình 1. 16: Giao diện chính Spyder

Khi bạn viết code xong ở phần viết code, ấn nút chạy chương trình (hoặc F5) thì bạn cần phải lưu file trước nếu file chưa được lưu.



Hình 1. 17: Lưu chương trình với đuôi *.py

Chọn thư mục để lưu vào viết tên file, tên file **luôn có .py đằng sau**, ví dụ như trong hình là **main.py**



Hình 1. 18: Chạy chương trình

CHƯƠNG 2. GIỚI THIỆU VỀ MẠNG NORON

2.1. Giới thiệu về Machine Learning

Những năm gần đây, AI - Artificial Intelligence (Trí Tuệ Nhân Tạo), và cụ thể hơn là Machine Learning (Học Máy hoặc Máy Học) nổi lên như một bằng chứng của cuộc cách mạng công nghiệp lần thứ tư (1 - động cơ hơi nước, 2 - năng lượng điện, 3 - công nghệ thông tin). Trí Tuệ Nhân Tạo đang len lỏi vào mọi lĩnh vực trong đời sống mà có thể chúng ta không nhận ra. Xe tự hành của Google và Tesla, hệ thống tự tag khuôn mặt trong ảnh của Facebook, trợ lý ảo Siri của Apple, hệ thống gợi ý sản phẩm của Amazon, hệ thống gợi ý phim của Netflix, máy chơi cờ vây AlphaGo của Google DeepMind, ..., chỉ là một vài trong vô vàn những ứng dụng của AI/Machine Learning.

Machine Learning là một tập con của AI. Theo định nghĩa của Wikipedia, *Machine learning is the subfield of computer science that “gives computers the ability to learn without being explicitly programmed”*. Nói đơn giản, Machine Learning là một lĩnh vực nhỏ của Khoa Học Máy Tính, nó có khả năng tự học hỏi dựa trên dữ liệu đưa vào mà không cần phải được lập trình cụ thể. Vậy Machine Learning là gì?

Machine learning gây nên cơn sốt công nghệ trên toàn thế giới trong vài năm nay. Trong giới học thuật, mỗi năm có hàng ngàn bài báo khoa học về đề tài này. Trong giới công nghiệp, từ các công ty lớn như Google, Facebook, Microsoft đến các công ty khởi nghiệp đều đầu tư vào machine learning. Hàng loạt các ứng dụng sử dụng machine learning ra đời trên mọi lĩnh vực của cuộc sống, từ khoa học máy tính đến những ngành ít liên quan hơn như vật lý, hóa học, y học, chính trị. *AlphaGo*, cỗ máy đánh cờ vây với khả năng tính toán trong một không gian có số lượng phần tử còn nhiều hơn số lượng hạt trong vũ trụ, tối ưu hơn bất kì đại kỉ thủ nào, là một trong rất nhiều ví dụ hùng hồn cho sự vượt trội của machine learning so với các phương pháp cổ điển

Để giới thiệu về machine learning, mình xin dựa vào mối quan hệ của nó với ba khái niệm sau:

- Machine learning và trí tuệ nhân tạo (Artificial Intelligence hay AI)
- Machine learning và Big Data.
- Machine learning và dự đoán tương lai.

Trí tuệ nhân tạo, AI, một cụm từ vừa gần gũi vừa xa lạ đối với chúng ta. Gần gũi bởi vì thế giới đang phát sốt với những công nghệ được dán nhãn AI. Xa lạ bởi vì một AI thực thụ vẫn còn nằm ngoài tầm với của chúng ta. Nói đến AI, hẳn mỗi người sẽ liên tưởng đến một hình ảnh khác nhau. Vài thập niên gần đây có một sự thay đổi về diện mạo của AI trong các bộ phim quốc

tế. Trước đây, các nhà sản xuất phim thường xuyên đưa hình ảnh robot vào phim (như *Terminator*), nhằm gieo vào đầu người xem suy nghĩ rằng trí tuệ nhân tạo là một phương thức nhân bản con người bằng máy móc. Tuy nhiên, trong những bộ phim gần hơn về đề tài này, ví dụ như *Transcendence* do Johnny Depp vào vai chính, ta không thấy hình ảnh của một con robot nào cả. Thay vào đó là một bộ não điện toán không lồ chỉ huy hàng vạn con Nanobot, được gọi là Singularity. Tất nhiên cả hai hình ảnh đều là hư cấu và giả tưởng, nhưng sự thay đổi như vậy cũng một phần nào phản ánh sự thay đổi ý niệm của con người về AI. AI bây giờ được xem như vô hình vô dạng, hay nói cách khác có thể mang bất cứ hình dạng nào. Vì nói về AI là nói về một *bộ não*, chứ không phải nói về một cơ thể, là software chứ không phải là hardware.

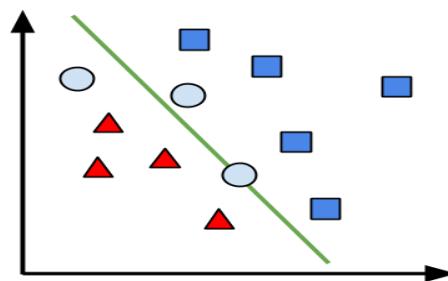
AI thể hiện một *mục tiêu* của con người. Machine learning là một *phương tiện* được kỳ vọng sẽ giúp con người đạt được mục tiêu đó. Và thực tế thì machine learning đã mang nhân loại đi rất xa trên quãng đường chinh phục AI. Nhưng vẫn còn một quãng đường xa hơn rất nhiều cần phải đi. Machine learning và AI có mối quan hệ chặt chẽ với nhau nhưng không hẳn là trùng khớp vì một bên là mục tiêu (AI), một bên là phương tiện (machine learning). Chinh phục AI mặc dù vẫn là mục đích tối thượng của machine learning, nhưng hiện tại machine learning tập trung vào những mục tiêu ngắn hạn hơn như: Làm cho máy tính có những khả năng nhận thức cơ bản của con người như nghe, nhìn, hiểu được ngôn ngữ, giải toán, lập trình, ...Và Hỗ trợ con người trong việc xử lý một khối lượng thông tin khổng lồ mà chúng ta phải đổi mới hàng ngày, hay còn gọi là Big Data.

Big Data thực chất không phải là một ngành khoa học chính thống. Đó là một cụm từ dân gian và được giới truyền thông tung hô để ám chỉ thời kì bùng nổ của dữ liệu hiện nay. Nó cũng không khác gì với những cụm từ như "cách mạng công nghiệp", "kỉ nguyên phần mềm". Big Data là một hệ quả tất yếu của việc mạng Internet ngày càng có nhiều kết nối. Với sự ra đời của các mạng xã hội như Facebook, Instagram, Twitter, nhu cầu chia sẻ thông tin của con người tăng trưởng một cách chóng mặt. Youtube cũng có thể được xem là một mạng xã hội, nơi mọi người chia sẻ video và comment về nội dung của video.

Bùng nổ thông tin không phải là lý do duy nhất dẫn đến sự ra đời của cụm từ Big Data. Nên nhớ rằng Big Data xuất hiện mới từ vài năm gần đây nhưng khối lượng dữ liệu tích tụ kể từ khi mạng Internet xuất hiện vào cuối thế kỉ trước cũng không phải là nhỏ. Thế nhưng, lúc ấy con người ngồi quanh một đống dữ liệu và không biết làm gì với chúng ngoài lưu trữ và sao chép. Cho đến một ngày, các nhà khoa học nhận ra rằng trong đống dữ liệu ấy thực ra chứa một khối lượng tri thức khổng lồ. Những tri thức ấy có thể giúp cho ta hiểu thêm về con người và xã hội. Từ danh sách bộ phim yêu thích của một cá nhân chúng ta có thể rút ra được sở thích của người đó và giới thiệu những bộ

phim người ấy chưa từng xem, nhưng phù hợp với sở thích. Từ danh sách tìm kiếm của cộng đồng mạng chúng ta sẽ biết được vấn đề nóng hổi nhất đang được quan tâm và sẽ tập trung đăng tải nhiều tin tức hơn về vấn đề đó. Big Data chỉ thực sự bắt đầu từ khi chúng ta hiểu được giá trị của thông tin ẩn chứa trong dữ liệu, và có đủ tài nguyên cũng như công nghệ để có thể khai thác chúng trên quy mô khổng lồ. Và không có gì ngạc nhiên khi machine learning chính là thành phần mấu chốt của công nghệ đó. Ở đây ta có một quan hệ hỗ tương giữa machine Learning và Big Data: machine learning phát triển hơn nhờ sự gia tăng của khối lượng dữ liệu của Big Data; ngược lại, giá trị của Big Data phụ thuộc vào khả năng khai thác tri thức từ dữ liệu của machine learning.

Ngược dòng lịch sử, machine learning đã xuất hiện từ rất lâu trước khi mạng Internet ra đời. Một trong những thuật toán machine learning đầu tiên là *thuật toán perceptron* được phát minh ra bởi Frank Rosenblatt vào năm 1957. Đây là một thuật toán kinh điển dùng để phân loại hai khái niệm. Một ví dụ đơn giản là phân loại thư rác (tam giác) và thư bình thường (vuông). Chắc các bạn sẽ khó hình dung ra được làm thế nào để làm được điều đó. Đối với perceptron, điều này không khác gì với việc vẽ một đường thẳng trên mặt phẳng để phân chia hai tập điểm:

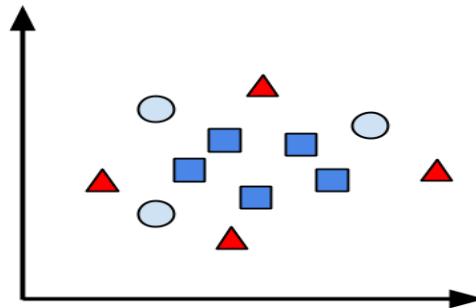


Hình 2.1. Đường thẳng trên mặt phẳng để phân chia hai tập điểm

Những điểm tam giác và vuông đại diện cho những email chúng ta đã biết nhãn trước. Chúng được dùng để "huấn luyện" (train) perceptron. Sau khi vẽ đường thẳng chia hai tập điểm, ta nhận thêm các điểm chưa được dán nhãn, đại diện cho các email cần được phân loại (điểm tròn). Ta dán nhãn của một điểm theo nhãn của các điểm cùng nửa mặt phẳng với điểm đó.

Sơ lược quy trình phân loại thư được mô tả sau. Trước hết, ta cần một thuật toán để chuyển email thành những điểm dữ liệu. Công đoạn này rất quan trọng vì nếu chúng ta chọn được biểu diễn phù hợp, công việc của perceptron sẽ nhẹ nhàng hơn rất nhiều. Tiếp theo, perceptron sẽ đọc tọa độ của từng điểm và sử dụng thông tin này để cập nhật tham số của đường thẳng cần tìm. Các bạn có thể xem qua demo của perceptron (điểm xanh lá cây là điểm perceptron đang xử lý)

Vì là một thuật toán khá đơn giản, có rất nhiều vấn đề có thể nảy sinh với perceptron, ví dụ như điểm cần phân loại nằm ngay trên đường thẳng phân chia. Hoặc tệ hơn là với một tập dữ liệu phức tạp hơn, đường thẳng phân chia không tồn tại:



Hình 2.2 Đường thẳng phân chia không tồn tại

Lúc này, ta cần các loại đường phân chia "không thẳng". Nhưng đó lại là một câu chuyện khác.

Perceptron là một thuật toán supervised learning: ta đưa cho máy tính hàng loạt các ví dụ cùng câu trả lời mẫu với hy vọng máy tính sẽ tìm được những đặc điểm cần thiết để đưa ra dự đoán cho những ví dụ khác chưa có câu trả lời trong tương lai. Ngoài ra, cũng có những thuật toán machine learning không cần câu trả lời mẫu, được gọi là unsupervised learning. Trong trường hợp này, máy tính cố gắng khai thác ra cấu trúc ẩn của một tập dữ liệu mà không cần câu trả lời mẫu. Một loại machine learning khác được gọi là reinforcement learning. Trong dạng này, cũng không hề có câu trả lời mẫu, nhưng thay vì đó máy tính nhận được phản hồi cho mỗi hành động. Dựa vào phản hồi tích cực hay tiêu cực mà máy tính sẽ điều chỉnh hoạt động cho phù hợp. Sau đây là một ví dụ minh họa

Mục tiêu của chiếc xe là leo lên được đỉnh đồi và lấy được ngôi sao. Chiếc xe có hai chuyển động tới và lui. Bằng cách thử các chuyển động và nhận được phản hồi là độ cao đạt được và thời gian để lấy được ngôi sao, chiếc xe dần trở nên thuận thực hơn trong việc leo đồi lấy sao.

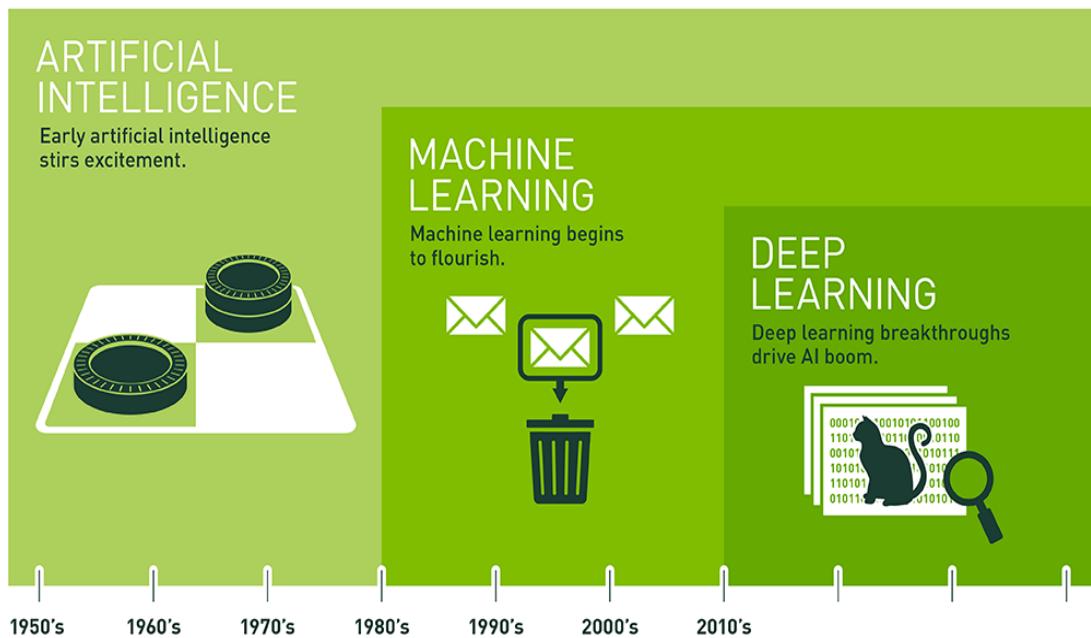
Machine learning có mối quan hệ rất mật thiết đối với statistics (thống kê). Machine learning sử dụng các mô hình thống kê để "ghi nhớ" lại sự phân bố của dữ liệu. Tuy nhiên, không đơn thuần là ghi nhớ, machine learning phải có khả năng tổng quát hóa những gì đã được nhìn thấy và đưa ra dự đoán cho những trường hợp chưa được nhìn thấy. Bạn có thể hình dung một mô hình machine learning không có khả năng tổng quát như một đứa trẻ học vẹt: chỉ trả lời được những câu trả lời mà nó đã học thuộc lòng đáp án. Khả năng tổng quát là một khả năng tự nhiên và kì diệu của con người: bạn không thể nhìn thấy tất cả các khuôn mặt người trên thế giới nhưng bạn có thể nhận biết được

một thứ có phải là khuôn mặt người hay không với xác suất đúng gần như tuyệt đối. Đỉnh cao của machine learning sẽ là mô phỏng được khả năng tổng quát hóa và suy luận này của con người.

Như ta đã thấy, nói đến machine learning là nói đến "dự đoán": từ việc dự đoán nhãn phân loại đến dự đoán hành động cần thực hiện trong bước tiếp theo. Vậy machine learning có thể dự đoán tương lai hay không? Có thể có hoặc có thể không: machine learning có thể dự đoán được tương lai, nhưng chỉ khi tương lai có mối liên hệ mật thiết với hiện tại.

Để kết thúc, mình muốn cùng các bạn xem xét một ví dụ đơn giản sau. Giả sử bạn được đưa cho một đồng xu, rồi được yêu cầu tung đồng xu một số lần. Vấn đề đặt ra là: dựa vào những lần tung đồng xu đó, bạn hãy tiên đoán ra kết quả lần tung tiếp theo. Chỉ cần dựa vào tỉ lệ sấp/ngửa của những lần tung trước đó, bạn có thể đưa ra một dự đoán khá tốt. Nhưng nếu mỗi lần tung, người ta đưa cho bạn một đồng xu khác nhau thì mọi chuyện sẽ hoàn toàn khác. Các đồng xu khác nhau có xác suất sấp ngửa khác nhau. Lúc này việc dự đoán gần như không thể vì xác suất sấp ngửa của lần tung sau không hề liên quan gì đến lần tung trước. Điều tương tự cũng xảy ra với việc dự đoán tương lai bằng machine learning, nếu ta xem như mỗi ngày có một "đồng xu" được tung ra để xem một sự kiện có diễn ra hay không. Nếu "đồng xu" của ngày mai được chọn một cách tùy ý không theo phân bố nào cả thì machine learning sẽ thất bại. Rất may là trong nhiều trường hợp điều đó không hoàn toàn đúng, thế giới hoạt động theo những quy luật nhất định và machine learning có thể nhận ra những quy luật đó. Nhưng nói cho cùng, machine learning hoàn toàn không phải là một bà phủ thủy với quả cầu tiên tri mà cũng giống như chúng ta: phán đoán bằng cách tổng quát hóa những kinh nghiệm, những gì đã được học từ dữ liệu.

Những năm gần đây, khi mà khả năng tính toán của các máy tính được nâng lên một tầm cao mới và lượng dữ liệu khổng lồ được thu thập bởi các hãng công nghệ lớn, Machine Learning đã tiến thêm một bước dài và một lĩnh vực mới được ra đời gọi là Deep Learning (Học Sâu - *thực sự tôi không muốn dịch từ này ra tiếng Việt*). Deep Learning đã giúp máy tính thực thi những việc tưởng chừng như không thể vào 10 năm trước: phân loại cả ngàn vật thể khác nhau trong các bức ảnh, tự tạo chú thích cho ảnh, bắt chước giọng nói và chữ viết của con người, giao tiếp với con người, hay thậm chí cả sáng tác văn hay âm nhạc



Since an early flush of optimism in the 1950s, smaller subsets of artificial intelligence – first machine learning, then deep learning, a subset of machine learning – have created ever larger disruptions.

Hình 2.3. Lịch sử phát triển của Maching Learning

2.2. Lịch sử phát triển mạng nơron nhân tạo- ANN

Các nghiên cứu về bộ não con người đã được tiến hành từ hàng nghìn năm nay. Cùng với sự phát triển của khoa học kỹ thuật đặc biệt là những tiến bộ trong ngành điện tử hiện đại, việc con người bắt đầu nghiên cứu các nơron nhân tạo là hoàn toàn tự nhiên. Có thể tính từ nghiên cứu của William (1890) về tâm lý học với sự liên kết các nơron thần kinh. Sự kiện đầu tiên đánh dấu sự ra đời của mạng nơron nhân tạo diễn ra vào năm 1943 khi nhà thần kinh học Warren McCulloch và nhà toán học Walter Pitts viết bài báo mô tả cách thức các nơron hoạt động. Họ cũng đã tiến hành xây dựng một mạng nơron đơn giản bằng các mạch điện. Các nơron của họ được xem như là các thiết bị nhị phân với ngưỡng cố định. Kết quả của các mô hình này là các hàm logic đơn giản chẳng hạn như “a OR b” hay “a AND b”. Những tiến bộ của máy tính đầu những năm 1950 giúp cho việc mô hình hóa các nguyên lý của những lý thuyết liên quan tới cách thức con người suy nghĩ đã trở thành hiện thực. Nathaniel Rochester sau nhiều năm làm việc tại các phòng thí nghiệm nghiên cứu của IBM đã có những nỗ lực đầu tiên để mô phỏng một mạng nơron.

Năm 1956 dự án Dartmouth nghiên cứu về trí tuệ nhân tạo (Artificial Intelligence) đã mở ra thời kỳ phát triển mới cả trong lĩnh vực trí tuệ nhân tạo lẫn mạng nơron. Tác động tích cực của nó là thúc đẩy hơn nữa sự quan tâm của các nhà khoa học về trí tuệ nhân tạo và quá trình xử lý ở mức đơn giản của mạng nơron trong bộ não con người.

Những năm tiếp theo của dự án Dartmouth, John von Neumann đã đề xuất việc mô phỏng các nơron đơn giản bằng cách sử dụng role điện áp hoặc đèn chân không. Nhà sinh học chuyên nghiên cứu về nơron Frank Rosenblatt cũng bắt đầu nghiên cứu về Perceptron năm 1958. Sau thời gian nghiên cứu này Perceptron đã được cài đặt trong phần cứng máy tính và được xem như là mạng nơron lâu đời nhất còn được sử dụng đến ngày nay. Perceptron một tầng rất hữu ích trong việc phân loại một tập các đầu vào có giá trị liên tục vào một trong hai lớp. Perceptron tính tổng có trọng số các đầu vào, rồi trừ tổng này cho một ngưỡng và cho ra một trong hai giá trị mong muốn có thể. Tuy nhiên Perceptron còn rất nhiều hạn chế, những hạn chế này đã được chỉ ra trong cuốn sách về Perceptron của Marvin Minsky và Seymour Papert của MIT (Massachusetts Institute of Technology) viết năm 1969 đã chứng minh nó không dùng được cho các hàm logic phức.

Năm 1959, Bernard Widrow và Marcian Hoff thuộc trường đại học Stanford đã xây dựng mô hình ADALINE (ADaptive LINear Elements) và MADALINE. (Multiple ADaptive LINear Elements). Các mô hình này sử dụng quy tắc học Least-Mean-Squares (LMS : Tối thiểu bình phương trung bình). MADALINE là mạng nơron đầu tiên được áp dụng để giải quyết một bài toán thực tế. Nó là một bộ lọc thích ứng có khả năng loại bỏ tín hiệu dội lại trên đường dây điện thoại. Ngày nay mạng nơron này vẫn được sử dụng trong các ứng dụng thương mại. Năm 1973 Von der Marlsburg: đưa ra quá trình học cạnh tranh và self – organization.

Năm 1974 Paul Werbos đã phát triển và ứng dụng phương pháp học lan truyền ngược (back-propagation). Tuy nhiên phải mất một vài năm thì phương pháp này mới trở lên phổ biến. Các mạng lan truyền ngược được biết đến nhiều nhất và được áp dụng rộng rãi nhất cho đến ngày nay.

Năm 1985, viện vật lý Hoa Kỳ bắt đầu tổ chức các cuộc họp hàng năm về mạng nơron ứng dụng trong tin học (Noron Networks for Computing).

Năm 1987, hội thảo quốc tế đầu tiên về mạng neuron của Viện các kỹ sư điện và điện tử IEEE (Institute of Electrical and Electronic Engineer) đã thu hút hơn 1800 người tham gia. Tính từ năm 1987 đến nay, hàng năm thế giới đều mở hội nghị toàn cầu chuyên ngành nơron IJCNN (International Joint Conference on Noron Networks).

Ngày nay, không chỉ dừng lại ở mức nghiên cứu lý thuyết, các nghiên cứu ứng dụng mạng nơron để giải quyết các bài toán thực tế được diễn ra ở khắp mọi nơi. Các ứng dụng mạng nơron ra đời ngày càng nhiều và ngày càng hoàn thiện hơn. Diễn hình là các ứng dụng: xử lý ngôn ngữ (Language Processing), nhận dạng ký tự (Character Recognition), nhận dạng tiếng nói

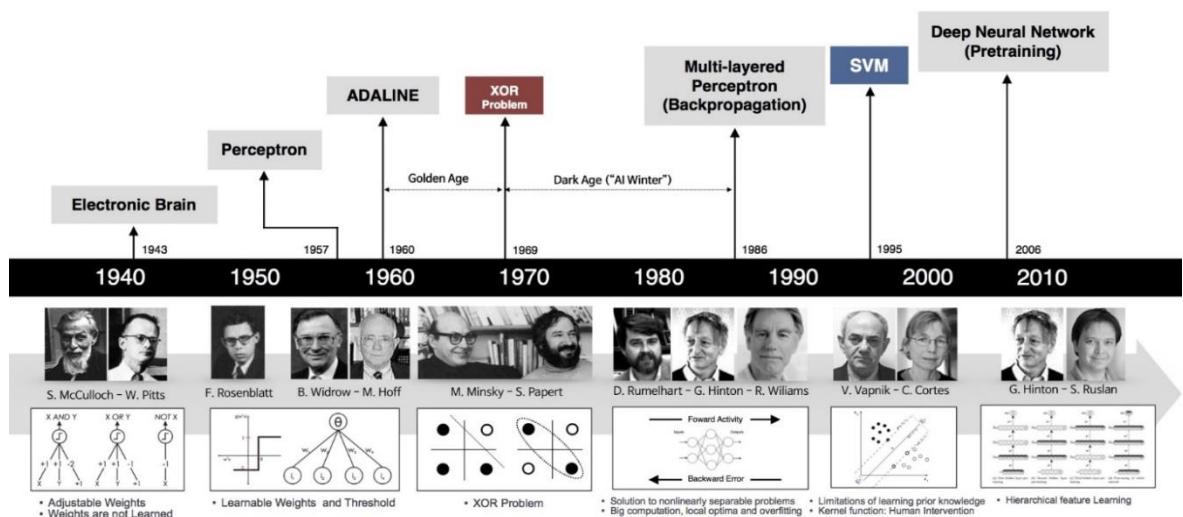
(Voice Recognition), nhận dạng mẫu (Pattern Recognition), xử lý tín hiệu (Signal Processing), Lọc dữ liệu (Data Filtering),...

2.3. Lịch sử phát triển Deep Learning

Trí tuệ nhân tạo đang len lỏi vào trong cuộc sống và ảnh hưởng sâu rộng tới mỗi chúng ta. Kể từ khi tôi viết bài đầu tiên, tàn suất chúng ta nghe thấy các cụm từ ‘artificial intelligence’, ‘machine learning’, ‘deep learning’ cũng ngày một tăng lên. Nguyên nhân chính dẫn đến việc này (và việc ra đời blog này) là sự xuất hiện của deep learning trong 5-6 năm gần đây.

Deep learning được nhắc đến nhiều trong những năm gần đây, nhưng những nền tảng cơ bản đã xuất hiện từ rất lâu ...

Chúng ta cùng quan sát hình dưới đây:



Hình 2.4. Lịch sử phát triển Deep Learning

- Perceptron (60s)

Một trong những nền móng đầu tiên của neural network và deep learning là perceptron learning algorithm (hoặc gọn là perceptron). Perceptron là một thuật toán supervised learning giúp giải quyết bài toán phân lớp nhị phân, được khởi nguồn bởi Frank Rosenblatt năm 1957 trong một nghiên cứu được tài trợ bởi Văn phòng nghiên cứu hải quân Hoa Kỳ (U.S Office of Naval Research – *từ một cơ quan liên quan đến quân sự*). Thuật toán perceptron được chứng minh là hội tụ nếu hai lớp dữ liệu là *linearly separable*. Với thành công này, năm 1958, trong một hội thảo, Rosenblatt đã có một phát biểu gây tranh cãi. Từ phát biểu này, tờ New York Times đã có một bài báo cho rằng perceptron được Hải quân Hoa Kỳ mong đợi “có thể đi, nói chuyện, nhìn, viết, tự sinh sản, và tự nhận thức được sự tồn tại của mình”. (*Chúng ta biết rằng cho tới giờ các hệ thống nâng cao hơn perceptron nhiều lần vẫn chưa thể*).

Mặc dù thuật toán này mang lại nhiều kỳ vọng, nó nhanh chóng được chứng minh không thể giải quyết những bài toán đơn giản. Năm 1969, Marvin Minsky và Seymour Papert trong cuốn sách nổi tiếng Perceptrons đã chứng minh rằng không thể ‘học’ được hàm số XOR khi sử dụng perceptron. Phát hiện này làm choáng váng giới khoa học thời gian đó. Perceptron được chứng minh rằng chỉ hoạt động nếu dữ liệu là *linearly separable*.

*Phát hiện này khiến cho các nghiên cứu về perceptron bị gián đoạn gần 20 năm. Thời kỳ này còn được gọi là **Mùa đông AI thứ nhất (The First AI winter)**. Cho tới khi...*

- MLP và Backpropagation ra đời (80s)

Geoffrey Hinton tốt nghiệp PhD ngành neural networks năm 1978. Năm 1986, ông cùng với hai tác giả khác xuất bản một bài báo khoa học trên Nature với tựa đề “Learning representations by back-propagating errors”. Trong bài báo này, nhóm của ông chứng minh rằng neural nets với nhiều hidden layer (được gọi là multi-layer perceptron hoặc MLP) có thể được huấn luyện một cách hiệu quả dựa trên một quy trình đơn giản được gọi là **backpropagation**. Việc này giúp neural nets *thoát* được những hạn chế của perceptron về việc chỉ biểu diễn được các quan hệ tuyến tính. Để biểu diễn các quan hệ phi tuyến, phía sau mỗi layer là một hàm kích hoạt phi tuyến, ví dụ hàm sigmoid hoặc tanh. (ReLU ra đời năm 2012). Với hidden layers, neural nets được chứng minh rằng có khả năng xấp xỉ hầu hết bất kỳ hàm số nào qua một định lý được gọi là universal approximation theorem.

Thuật toán này mang lại một vài thành công ban đầu, nổi trội là **convolutional neural nets** (convnets hay CNN) (còn được gọi là LeNet) cho bài toán nhận dạng chữ số viết tay được khởi nguồn bởi Yann LeCun tại AT&T Bell Labs (Yann LeCun là sinh viên sau cao học của Hinton tại đại học Toronto năm 1987-1988). Dưới đây là bản demo được lấy từ trang web của LeNet, network là một CNN với 5 layer, còn được gọi là LeNet-5 (1998).

- Mùa đông AI thứ hai (90s - đầu 2000s)

Các mô hình tương tự được kỳ vọng sẽ giải quyết nhiều bài toán image classification khác. Tuy nhiên, không như các chữ số, các loại ảnh khác lại rất hạn chế vì máy ảnh số chưa phổ biến tại thời điểm đó. Ảnh được gán nhãn lại càng hiếm. Trong khi để có thể huấn luyện được mô hình convnets, ta cần rất nhiều dữ liệu huấn luyện. Ngay cả khi dữ liệu có đủ, một vấn đề nan giải khác là khả năng tính toán của các máy tính thời đó còn rất hạn chế.

Một hạn chế khác của các kiến trúc MLP nói chung là hàm mất mát không phải là một hàm lồi. Việc này khiến cho việc tìm nghiệm tối ưu toàn cục cho bài toán tối ưu hàm mất mát trở nên rất khó khăn. Một vấn đề khác liên quan đến giới hạn tính toán của máy tính cũng khiến cho việc huấn luyện

MLP không hiệu quả khi số lượng hidden layers lớn lên. Vấn đề này có tên là **vanishing gradient**.

Nhắc lại rằng hàm kích hoạt được sử dụng thời gian đó là sigmoid hoặc tanh – là các hàm bị chặn trong khoảng $(0, 1)$ hoặc $(-1, 1)$ (Nhắc lại đạo hàm của hàm sigmoid $\sigma(z)\sigma'(z) = \sigma(z)(1-\sigma(z))\sigma'(z)(1-\sigma(z))$ là tích của hai số nhỏ hơn 1). Khi sử dụng backpropagation để tính đạo hàm cho các ma trận hệ số ở các lớp đầu tiên, ta cần phải nhân rất nhiều các giá trị nhỏ hơn 1 với nhau. Việc này khiến cho nhiều đạo hàm thành phần bằng 0 do xấp xỉ tính toán. Khi đạo hàm của một thành phần bằng 0, nó sẽ không được cập nhật thông qua gradient descent!

Những hạn chế này khiến cho neural nets một lần nữa rơi vào thời kỳ *băng giá*. Vào thời điểm những năm 1990 và đầu những năm 2000, neural nets dần được thay thế bởi support vector machines –SVM. SVMs có ưu điểm là bài toán tối ưu để tìm các tham số của nó là một bài toán lồi – có nhiều các thuật toán tối ưu hiệu quả giúp tìm nghiệm của nó. Các kỹ thuật về kernel cũng phát triển giúp SVMs giải quyết được cả các vấn đề về việc dữ liệu không phân biệt tuyến tính.

Nhiều nhà khoa học làm machine learning chuyển sang nghiên cứu SVM trong thời gian đó, trừ một vài nhà khoa học cứng đầu...

- Cái tên được làm mới – Deep Learning (2006)

Năm 2006, Hinton một lần nữa cho rằng ông biết bộ não hoạt động như thế nào, và giới thiệu ý tưởng của *tiền huấn luyện không giám sát* (*unsupervised pretraining*) thông qua deep belief nets (DBN). DBN có thể được xem như sự xếp chồng các unsupervised networks đơn giản như restricted Boltzman machine hay autoencoders.

Lấy ví dụ với autoencoder. Mỗi autoencoder là một neural net với một hidden layer. Số hidden unit ít hơn số input unit, và số output unit bằng với số input unit. Network này đơn giản được huấn luyện để kết quả ở output layer giống với kết quả ở input layer (và vì vậy được gọi là autoencoder). Quá trình dữ liệu đi từ input layer tới hidden layer có thể coi là *mã hóa*, quá trình dữ liệu đi từ hidden layer ra output layer có thể được coi là *giải mã*. Khi output giống với input, ta có thể thấy rằng hidden layer với ít unit hơn có thể mã hóa input khá thành công, và có thể được coi mang những tính chất của input. Nếu ta bỏ output layer, cố định (*freeze*) kết nối giữa input và hidden layer, coi đầu ra của hidden layer là một input mới, sau đó huấn luyện một autoencoder khác, ta được thêm một hidden layer nữa. Quá trình này tiếp tục kéo dài ta sẽ được một network đủ sâu mà output của network lớn này (chính là hidden layer của autoencoder cuối cùng) mang thông tin của input ban đầu. Sau đó ta có thể thêm các layer khác tùy thuộc vào bài toán (chẳng hạn thêm softmax layer ở

cuối cho bài toán classification). Cả network được huấn luyện thêm một vài epoch nữa. Quá trình này được gọi là *tinh chỉnh* (*fine tuning*).

Tại sao quá trình huấn luyện như trên mang lại nhiều lợi ích?

Một trong những hạn chế đã đề cập của MLP là vấn đề *vanishing gradient*. Những ma trận trọng số ứng với các layer đầu của network rất khó được huấn luyện vì đạo hàm của hàm mất mát theo các ma trận này nhỏ. Với ý tưởng của DBN, các ma trận trọng số ở những hidden layer đầu tiên được *tiền huấn luyện* (*pretrained*). Các trọng số được tiền huấn luyện này có thể coi là giá trị khởi tạo tốt cho các hidden layer phía sau. Việc này giúp phần nào tránh được sự phiền hà của *vanishing gradient*.

Kể từ đây, neural networks với nhiều hidden layer được đổi tên thành **deep learning**.

Vấn đề *vanishing gradient* được giải quyết phần nào (vẫn chưa thực sự triệt để), nhưng vẫn còn những vấn đề khác của deep learning: dữ liệu huấn luyện quá ít, và khả năng tính toán của CPU còn rất hạn chế trong việc huấn luyện các deep networks.

Năm 2010, giáo sư Fei-Fei Li, một giáo sư ngành computer vision đầu ngành tại Stanford, cùng với nhóm của bà tạo ra một cơ sở dữ liệu có tên ImageNet với hàng triệu bức ảnh thuộc 1000 lớp dữ liệu khác nhau đã được gán nhãn. Dự án này được thực hiện nhờ vào sự bùng nổ của internet những năm 2000 và lượng ảnh khổng lồ được upload lên internet thời gian đó.

Bộ cơ sở dữ liệu này được cập nhật hàng năm, và kể từ năm 2010, nó được dùng trong một cuộc thi thường niên có tên ImageNet Large Scale Visual Recognition Challenge (ILSVRC). Trong cuộc thi này, dữ liệu huấn luyện được giao cho các đội tham gia. Mỗi đội cần sử dụng dữ liệu này để huấn luyện các mô hình phân lớp, các mô hình này sẽ được áp dụng để dự đoán nhãn của dữ liệu mới (được giữ bởi ban tổ chức). Trong hai năm 2010 và 2011, có rất nhiều đội tham gia. Các mô hình trong hai năm này chủ yếu là sự kết hợp của SVM với các feature được xây dựng bởi các bộ *hand-crafted descriptors* (SIFT, HoG, v.v.). Mô hình giành chiến thắng có top-5 error rate là 28% (càng nhỏ càng tốt). Mô hình giành chiến thắng năm 2011 có top-5 error rate là 26%. Cải thiện không nhiều!

- Đột phá (2012)

Năm 2012, cũng tại ILSVRC, Alex Krizhevsky, Ilya Sutskever, và Geoffrey Hinton (lại là ông) tham gia và đạt kết quả top-5 error rate 16%. Kết quả này làm sững sờ giới nghiên cứu thời gian đó. Mô hình là một Deep Convolutional Neural Network, sau này được gọi là AlexNet.

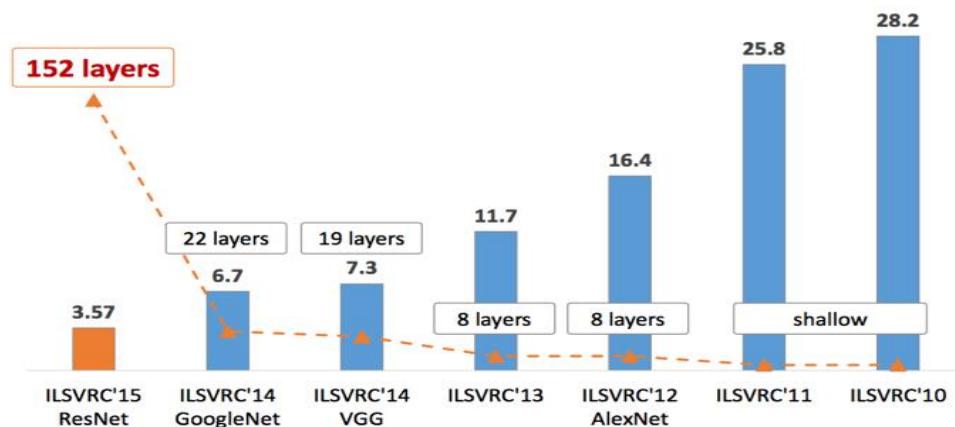
Trong bài báo này, rất nhiều các kỹ thuật mới được giới thiệu. Trong đó hai đóng góp nổi bật nhất là hàm ReLU và dropout. Hàm ReLU

($\text{ReLU}(x) = \max(x, 0)$) với cách tính và đạo hàm đơn giản (bằng 1 khi đầu vào không âm, bằng 0 khi ngược lại) giúp tốc độ huấn luyện tăng lên đáng kể. Ngoài ra, việc ReLU không bị chặn trên bởi 1 (như softmax hay tanh) khiến cho vấn đề vanishing gradient cũng được giải quyết phần nào. Dropout cũng là một kỹ thuật đơn giản và cực kỳ hiệu quả. Trong quá trình training, nhiều hidden unit bị tắt ngẫu nhiên và mô hình được huấn luyện trên các bộ tham số còn lại. Trong quá trình test, toàn bộ các unit sẽ được sử dụng. Cách làm này khá là có lý khi đối chiếu với con người. Nếu chỉ dùng một phần năng lực đã đem lại hiệu quả thì dùng toàn bộ năng lực sẽ mang lại hiệu quả cao hơn. Việc này cũng giúp cho mô hình tránh được overfitting và cũng được coi giống với kỹ thuật *ensemble* trong các hệ thống machine learning khác. Với mỗi cách tắt các unit, ta có một mô hình khác nhau. Với nhiều tổ hợp unit bị tắt khác nhau, ta thu được nhiều mô hình. Việc kết hợp ở cuối cùng được coi như sự kết hợp của nhiều mô hình (và vì vậy, nó giống với *ensemble learning*).

Một trong những yếu tố quan trọng nhất giúp AlexNet thành công là việc sử dụng GPU (card đồ họa) để huấn luyện mô hình. GPU được tạo ra cho game thủ, với khả năng chạy song song nhiều lõi, đã trở thành một công cụ cực kỳ phù hợp với các thuật toán deep learning, giúp tăng tốc thuật toán lên nhiều lần so với CPU.

Sau AlexNet, tất cả các mô hình giành giải cao trong các năm tiếp theo đều là các deep networks (ZFNet 2013, GoogLeNet 2014, VGG 2014, ResNet 2015). Tôi sẽ giành một bài của blog để viết về các kiến trúc quan trọng này. Xu thế chung có thể thấy là các mô hình càng ngày càng *deep*. Xem hình dưới đây.

Những công ty công nghệ lớn cũng để ý tới việc phát triển các phòng nghiên cứu deep learning trong thời gian này. Rất nhiều các ứng dụng công nghệ đột phá đã được áp dụng vào cuộc sống hàng ngày. Cũng kể từ năm 2012, số lượng các bài báo khoa học về deep learning tăng lên theo hàm số mũ. Các blog về deep learning cũng tăng lên từng ngày.



Hình 2.5. Kết quả ILSVRC qua các năm. (Nguồn: CNN Architectures: LeNet, AlexNet, VGG, GoogLeNet, ResNet and more ...)

2.4. Một số thư viện Deep Learning nổi tiếng Hiện nay

Cùng với sự phát triển của các thuật toán Deep Learning thì các thư viện cũng như framework hỗ trợ các thuật toán này cũng ngày càng tăng về số lượng. Hầu hết các thư viện và framework này đều cung cấp dưới dạng mã nguồn mở do đó rất linh hoạt trong việc sử dụng và mở rộng, đây cũng là một trong những lý do DL được áp dụng trong nhiều bài toán với nhiều lĩnh vực khác nhau. Trong phần tiếp theo nội dung post này sẽ giới thiệu một số thư viện phổ biến đang được cộng đồng nghiên cứu sử dụng.



Hình 2.6. Một số thư viện Deep learning nổi tiếng

- TensorFlow. (Commits: 16785, Contributors: 795)

Do các developer của Google phát triển, TensorFlow là thư viện nguồn mở của graphs computations thuộc luồng dữ liệu, thích hợp với Machine Learning. TensorFlow đáp ứng các requirement cao cấp trong môi trường Google để train Neural Networks và thư viện kế nhiệm của DistBelief – 1 hệ thống Machine Learning dựa trên Neural Networks. Tuy nhiên, TensorFlow không chỉ sử dụng cho mục đích khoa học trong Google mà có thể áp dụng trong các dự án thực tế.

Tính năng quan trọng của TensorFlow là hệ thống nút đa layer, cho phép huấn luyện các neural networks trên datasets lớn 1 cách nhanh chóng, hỗ trợ khả năng nhận diện giọng nói và định vị vật thể trong ảnh của Google.

Một số tính năng nổi bật của Torch framework:

- + TensorFlow hỗ trợ cả hai ngôn ngữ C và Python.
- + TensorFlow có thể chạy trên nhiều CPU cũng như GPU giúp đẩy nhanh quá trình huấn luyện cũng như xử lý dữ liệu thực từ mô hình đã được học. Ngoài ra với việc có thể sử dụng thư viện này trên các hệ thống cloud sẽ làm đẩy nhanh hiệu năng của các hệ thống sử dụng TensorFlow.

- + Với khả năng chạy trên nhiều hệ điều hành như bao gồm cả iOS, Android, hứa hẹn sẽ phát triển được các ứng dụng thông minh nhờ áp dụng các tính năng nổi bật của DL.
- Theano. (Commits: 25870, Contributors: 300)

Theano là package Python định dạng các arrays đa chiều tương tự như NumPy, đi kèm với các operation về toán và expressions. Thư viện này được compiled, chạy hiệu quả trên tất cả các architectures. Do đội ngũ Machine Learning của Université de Montréal, Theano được sử dụng chính cho các hoạt động liên quan đến Machine Learning.

Lưu ý là Theano tích hợp với NumPy ở mức độ operation cấp thấp. Thư viện này cũng tối ưu hóa khả năng sử dụng GPU & CPU, giúp cho hiệu năng của computation thiêng về data nhanh chóng hơn.

Hiệu quả và sự ổn định cũng mang đến những kết quả chính xác hơn, dù đó là những giá trị rất nhỏ như computation của $\log(1+x)$ sẽ cho ra kết quả chính xác đối với các giá trị nhỏ nhất của x.

Theano sử dụng 2 gói thư viện để hỗ trợ cho việc định nghĩa mô hình mạng neural:

- + **Lasagne**: Là thư viện định nghĩa các lớp (trừ lớp cuối cùng) của mô hình mạng neural. Lasagne giúp người dùng lưu trữ dữ liệu trong mạng, tính toán giá trị hàm lỗi, cập nhật trọng số.
- + **Keras**: Là lớp cuối cùng trong cấu trúc mạng. Hỗ trợ cài đặt các hàm kích hoạt và định nghĩa lớp softmax.
- Keras. (Commits: 3519, Contributors: 428)

Keras là thư viện nguồn mở được viết bằng Python dùng để build các Neural Networks ở cấp độ cao cấp của interface. Thư viện này đơn giản và có khả năng mở rộng cao. Keras sử dụng backend là Theano hoặc TensorFlow nhưng gần đây, Microsoft đã có gắng tích hợp CNTK (Cognitive Toolkit của Microsoft) thành back-end mới.

Cách tiếp cận đơn giản về thiết kế nhằm đến quy trình experimentation dễ dàng, nhanh chóng từ việc build các compact systems.

Bắt đầu dùng Keras rất đơn giản, tiếp theo là prototyping nhanh. Keras được viết bằng Python, theo mô-đun và mở rộng được. Không chỉ đơn giản và có tính định hướng cao, Keras vẫn hỗ trợ modeling rất mạnh mẽ.

Ý tưởng chung về Keras là dựa trên các layers và mọi thứ khác cũng đều được xây dựng xung quanh các layer này. Data được chuẩn bị trong các tensors, layer đầu tiên chịu trách nhiệm về input của các tensors, layer cuối cùng chịu trách nhiệm output và model được build ở giữa.

- SciKit-Learn (Commits: 21793, Contributors: 842)

Scikits là các packages bổ sung của SciPy Stack được thiết kế cho các chức năng chuyên biệt như xử lý ảnh và hỗ trợ Machine Learning. Riêng với mảng Machine Learning, một trong những ưu điểm nổi bật của các packages này là scikit-learn. Package được xây dựng trên nền tảng của SciPy và tận dụng các operations về toán.

Scikit-learn có giao diện đơn giản, nhất quán, exposes a concise and consistent interface to the common machine learning algorithms, hỗ trợ việc mang Machine Learning vào các hệ thống production trở nên đơn giản hơn. Thư viện này bao gồm các code chất lượng và documentation hay, dễ sử dụng, hiệu suất cao, là chuẩn mực thực tế cho xây dựng Machine Learning bằng Python.

- Matplotlib (Commits: 21754, Contributors: 588)

Một core package của SciPy Stack và 1 thư viện Python khác được xây dựng riêng cho việc generation các visualizations mạnh mẽ, đơn giản là Matplotlib. Matplotlib là 1 phần của phần mềm giúp cho Python (cùng với sự hỗ trợ của NumPy, SciPy và Pandas) trở thành đối thủ nổi bật với các công cụ khoa học như MatLab hoặc Mathematica.

Tuy nhiên, thư viện này ở cấp độ thấp, đồng nghĩa là bạn sẽ cần phải viết nhiều code hơn để tiếp cận các cấp độ visualization cao cấp và bạn sẽ phải nỗ lực hơn so với khi sử dụng các công cụ cấp cao, tuy nhiên nỗ lực này là hoàn toàn xứng đáng.

Chỉ cần nỗ lực 1 chút, bạn có thể tạo được các visualization bất kì:

- + Line plots;
- + Scatter plots;
- + Bar charts và Histograms;
- + Pie charts;
- + Stem plots;
- + Contour plots;
- + Quiver plots;
- + Spectrograms.

Có rất nhiều công cụ để tạo nhãn, lưới, các biểu tượng/ ký hiệu/ chú giải và rất nhiều yếu tố format khác với Matplotlib. Về cơ bản, mọi thứ đều có thể custom được.

Thư viện này còn được rất nhiều platform hỗ trợ và tận dụng các GUI kít khác nhau để mô tả các visualizations kết quả. Thay đổi các IDEs (như IPython) sẽ hỗ trợ chức năng của Matplotlib.

- SciPy (Commits: 17213, Contributors: 489)

SciPy là 1 thư viện phần mềm cho engineering và khoa học. Một lần nữa bạn cần phải hiểu sự khác biệt giữa SciPy Stack và thư viện SciPy. SciPy gồm các modules cho đại số tuyến tính, optimization, tích hợp và thống kê. Chức năng chính của thư viện SciPy được xây dựng trên NumPy, và arrays của nó sẽ tận dụng tối đa NumPy. Nó mang đến rất nhiều hoạt động hữu ích liên quan đến số như tích hợp số, optimization... qua các submodules chuyên biệt. Các hàm trong tất cả các submodules của SciPy đều được document tốt.

- NumPy (Commits: 15980, Contributors: 522)

Khi bắt đầu giải quyết task về khoa học bằng Python, tập hợp phần mềm được thiết kế riêng cho scientific computing trong Python sẽ không thể không hỗ trợ SciPy Stack của Python (đừng nhầm lẫn với thư viện SciPy – là 1 phần của stack này, và cộng đồng của stack này). Tuy nhiên, stack này khá rộng, có hơn cả tá thư viện trong nó và chúng ta thì lại muốn tập trung vào các core packages (đặc biệt là những packages quan trọng nhất).

Package cơ bản nhất, khi computation stack về khoa học được xây dựng là NumPy (viết tắt của Numerical Python), cung cấp rất nhiều tính năng hữu ích cho các phần operations trong n-arrays & matrices trong Python. Thư viện này cung cấp khả năng vector hóa các vận hành về toán trong type array NumPy, giúp cải thiện hiệu suất và theo đó là tốc độ execution.

- Pandas (Commits: 15089, Contributors: 762)

Pandas là 1 package Python được thiết kế để làm việc với dữ liệu đơn giản, trực quan, được “gắn nhãn” và có liên hệ với nhau. Pandas là công cụ hoàn hảo để tinh chỉnh và làm sạch dữ liệu. Pandas được thiết kế hỗ trợ cho các thao tác, tập hợp và visualize dữ liệu.

Có 2 data structure chính trong thư viện này:

“Series”—1 chiều

Series	
A	X0
B	X1
C	X2
D	X3

“Data Frames”, 2 chiều

DataFrame				
	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3

- ☞ Dễ dàng xóa và thêm cột từ DataFrame
 - ☞ Chuyển data structures đến các objects DataFrame
 - ☞ Xử lý các data bị mất, như NaNs
 - ☞ Khả năng bhóm lại theo chức năng
- Pytorch

PyTorch là một framework được xây dựng dựa trên python cung cấp nền tảng tính toán khoa học phục vụ lĩnh vực Deep learning. Pytorch tập trung vào 2 khả năng chính:

- ☞ Một sự thay thế cho bộ thư viện numpy để tận dụng sức mạnh tính toán của GPU.
- ☞ Một platform Deep learning phục vụ trong nghiên cứu, mang lại sự linh hoạt và tốc độ.

Ngoài ra còn có các thư viện khác cho DeepLearning như Neural Networks Zoo, Caffe, MXNet, Chainer (mới), Caffe2,

2.5. Các khái niệm trong mạng Neron

Mạng Neuron nhân tạo (Artificial Neural Network- ANN) là mô hình xử lý thông tin được mô phỏng dựa trên hoạt động của hệ thống thần kinh của sinh vật, bao gồm số lượng lớn các Neuron được gắn kết để xử lý thông tin. ANN giống như bộ não con người, được học bởi kinh nghiệm (thông qua huấn luyện), có khả năng lưu giữ những kinh nghiệm hiểu biết (tri thức) và sử dụng những tri thức đó trong việc dự đoán các dữ liệu chưa biết (unseen data).

Các ứng dụng của mạng Neuron được sử dụng trong rất nhiều lĩnh vực như điện, điện tử, kinh tế, quân sự,... để giải quyết các bài toán có độ phức tạp và đòi hỏi có độ chính xác cao như điều khiển tự động, khai phá dữ liệu, nhận dạng,...

2.5.1. Epoch

Một Epoch là khi tất cả dữ liệu được đưa vào mạng neural network 1 lần. Khi dữ liệu quá lớn, chúng ta không thể đưa hết mỗi lần 1 epoch để huấn luyện. Buộc lòng chúng ta phải chia nhỏ 1 epoch ra thành các batchs nhỏ hơn.

2.5.2. Batch Size

Batch size là số lượng mẫu dữ liệu trong một batch. Ở đây, khái niệm batch size và số lượng batch(number of batch) là hoàn toàn khác nhau.

Chúng ta không thể đưa hết toàn bộ dữ liệu vào huấn luyện trong 1 epoch, vì vậy chúng ta cần phải chia tập dữ liệu thành các phần (number of batch), mỗi phần có kích thước là batch size.

2.5.3. Iterations

Iterations là số lượng batches cần để hoàn thành 1 epoch. Ví dụ chúng ta có tập dữ liệu có 20,000 mẫu, batch size là 500, vậy chúng ta cần 50 lần lặp (iteration) để hoàn thành 1 epoch.

2.5.4. Loss Function

Kí hiệu là LLL, là thành phần cốt lõi của evaluation function và objective function. Cụ thể, trong công thức thường gặp:

$$\mathcal{L}_D(f_w) = \frac{1}{|D|} \sum_{(x,y) \in D} L(f_w(x), y)$$

thì hàm LLL chính là loss function.

Loss function trả về một số thực không âm thể hiện sự chênh lệch giữa hai đại lượng: y^{\wedge} , label được dự đoán và y , label đúng. Loss function giống như một hình thức để bắt model đóng phạt mỗi lần nó dự đoán sai, và số mức phạt tỉ lệ thuận với độ trầm trọng của sai sót. Trong mọi bài toán supervised learning, mục tiêu của ta luôn bao gồm giảm thiểu tổng mức phạt phải đóng. Trong trường hợp lý tưởng $y^{\wedge}=y$. loss function sẽ trả về giá trị cực tiểu bằng 0.

2.5.5. Momentum

Là một kỹ thuật đơn giản thường cải thiện cả tốc độ đào tạo và độ chính xác. Đào tạo một mạng nơ-ron là quá trình tìm các giá trị cho các trọng số và độ lệch sao cho đối với một tập hợp các giá trị đầu vào nhất định, các giá trị đầu ra được tính toán khớp với các giá trị đích đã biết, đúng, đã biết. Ví dụ: giả sử bạn đang cố gắng dự đoán các loài hoa iris (setosa, Versicolor hoặc virginica) dựa trên chiều dài sepal của hoa, chiều rộng của cánh hoa, chiều dài cánh hoa và chiều rộng cánh hoa.

2.5.6. Dataset

Là Tập dữ liệu huấn luyện trong Machine Learning; là bộ dữ liệu thực tế được sử dụng để huấn luyện mô hình để thực hiện các hành động khác nhau. Đây là dữ liệu thực tế mà các mô hình quy trình phát triển đang diễn ra học với nhiều thuật toán và API khác nhau để đào tạo máy hoạt động tự động

2.5.7. Learning rate

Là một siêu tham số có thể định cấu hình được sử dụng trong việc đào tạo các mạng thần kinh có giá trị dương nhỏ, thường nằm trong khoảng giữa

0,0 và 1,0. Tốc độ học tập kiểm soát mức độ nhanh chóng của mô hình thích ứng với vấn đề.

Learning rate có thể là siêu tham số quan trọng nhất khi định cấu hình mạng thần kinh của bạn. Do đó, điều quan trọng là phải biết cách điều tra các tác động của tốc độ học tập đối với hiệu suất của mô hình và xây dựng một trực giác về động lực của tốc độ học tập đối với hành vi của mô hình

2.5.8. Traing set

Là một tập dữ liệu có kích thước lớn, được dùng để training trong quá trình huấn luyện máy học. Đây chính là tập dữ liệu máy dùng để học và rút trích được những đặc điểm quan trọng để ghi nhớ lại. Tập training set sẽ gồm 2 phần:

- Input: sẽ là những dữ liệu đầu vào.
- Output: sẽ là những kết quả tương ứng với tập input.

Training set là tập các cặp input và output dùng để huấn luyện trong quá trình máy học.

2.5.9. Testing set

Testing set là tập dữ liệu dùng để test sau khi máy đã học xong. Một mô hình máy học sau khi được huấn luyện, sẽ cần phải được kiểm chứng xem nó có đạt hiệu quả không. Testing set chỉ gồm các giá trị input mà không có các giá trị output. Máy tính sẽ nhận những giá trị input này, và xử lý các giá trị, sau đó đưa ra output tương ứng cho giá trị input.

Testing set là tập các giá trị input và được dùng để kiểm thử độ chính xác của những mô hình máy học sau khi được huấn luyện.

CHƯƠNG 3. GIỚI THIỆU VỀ THU VIỆN PYTORCH

3.1. Lịch sử phát triển và các phiên bản Pytorch

PyTorch đã được phát hành vào năm 2016. Nhiều nhà nghiên cứu sẵn sàng áp dụng PyTorch ngày càng nhiều. Nó được vận hành bởi Facebook. Facebook cũng vận hành Caffe2 (Kiến trúc chuyển đổi để nhúng tính năng nhanh). Việc chuyển đổi một mô hình do PyTorch xác định thành Caffe2 là một thách thức. Với mục đích này, Facebook và Microsoft đã phát minh ra Sàn giao dịch mạng thần kinh mở (ONNX) vào tháng 9/2017. Nói một cách đơn giản, ONNX được phát triển để chuyển đổi các mô hình giữa các khung. Caffe2 đã được sáp nhập vào tháng 3 năm 2018 thành PyTorch.

PyTorch giúp dễ dàng xây dựng một mạng nơ-ron cực kỳ phức tạp. Tính năng này đã nhanh chóng biến nó thành một thư viện đi đến. Trong công việc nghiên cứu, nó mang đến một sự cạnh tranh gay gắt với TensorFlow. Các nhà phát minh của PyTorch muốn tạo ra một thư viện cực kỳ cần thiết, có thể dễ dàng chạy tất cả các tính toán số, và cuối cùng, họ đã phát minh ra PyTorch. Có một thách thức lớn đối với nhà khoa học Deep learning, nhà phát triển Machine learning và trình gỡ lỗi Mạng nơ-ron để chạy và kiểm tra một phần mã trong thời gian thực. PyTorch hoàn thành thử thách này và cho phép họ chạy và kiểm tra mã của họ trong thời gian thực. Vì vậy, họ không phải chờ đợi để kiểm tra xem nó có hoạt động hay không.

Các phiên bản:

- [master \(unstable\)](#)
- [v1.2.0 \(latest stable release\)](#)
- [v1.1.0](#)
- [v1.0.1](#)
- [v1.0.0](#)
- [v0.4.1](#)
- [v0.4.0](#)
- [v0.3.1](#)
- [v0.3.0](#)
- [v0.2.0](#)
- [v0.1.12](#)

3.2. Đặc điểm của Pytorch

➤ Ưu điểm:

- ❖ Mang lại khả năng debug dễ dàng hơn theo hướng interactively, rất nhiều nhà nghiên cứu và engineer đã dùng cả pytorch và tensorflow đều đánh giá cao pytorch hơn trong vấn đề debug và visualize.
- ❖ Hỗ trợ tốt dynamic graphs.
- ❖ Được phát triển bởi đội ngũ Facebook.

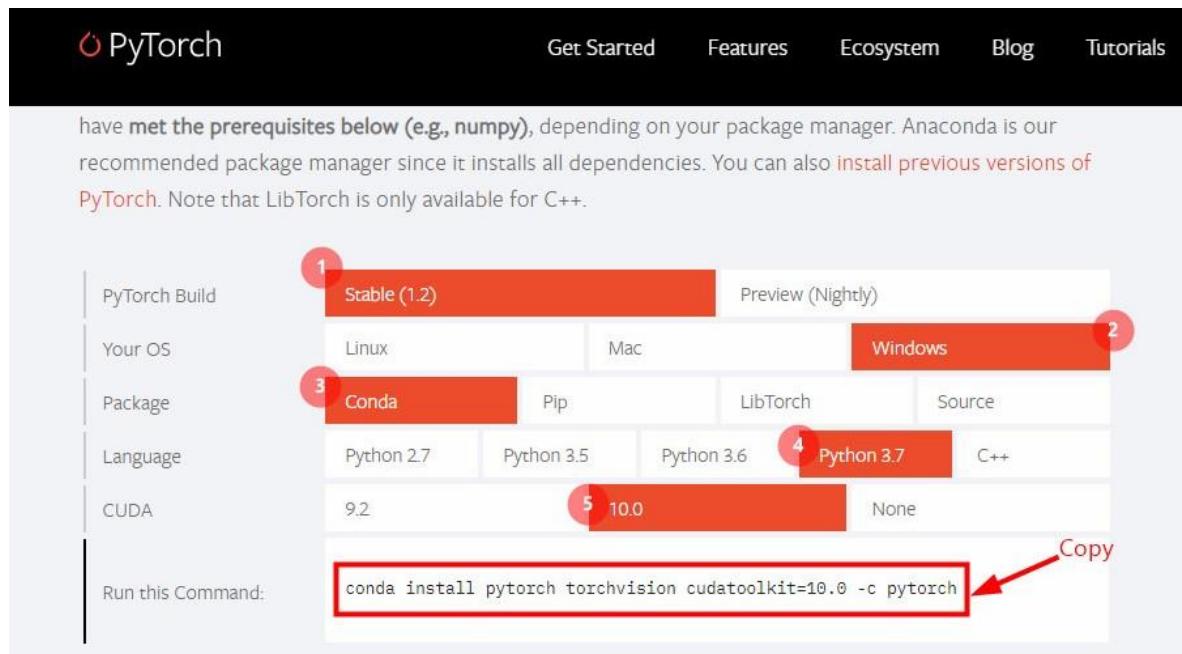
- ❖ Kết hợp cả các API cấp cao và cấp thấp.
- Nhược điểm:
 - ❖ Vẫn chưa được hoàn thiện trong việc deploy, áp dụng cho các hệ thống lớn,... được như framework ra đời trước nó như tensorflow.
 - ❖ Ngoài document chính từ pytorch thì vẫn còn khá hạn chế các nguồn tài liệu bên ngoài như các tutorials hay các câu hỏi trên stackoverflow.

3.3. Cài đặt thư viện Pytorch

Để cài đặt, đầu tiên, bạn phải chọn tùy chọn của bạn và sau đó chạy lệnh cài đặt. Bạn có thể bắt đầu cài đặt cục bộ hoặc với một đối tác đám mây. Trong sơ đồ bên dưới, Stable hiển thị phiên bản PyTorch (1.2) được hỗ trợ và thử nghiệm nhiều nhất hiện nay, phù hợp với nhiều người dùng. Nếu bạn muốn các bản dựng 1.2 mới nhất nhưng chưa được kiểm tra và hỗ trợ đầy đủ, thì bạn phải chọn Xem trước.

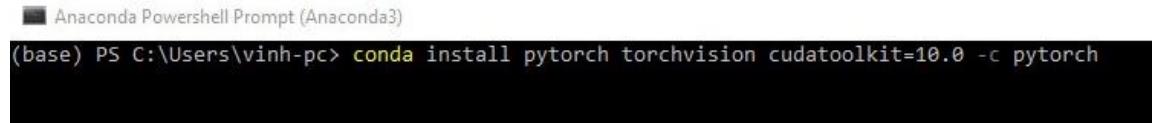
Để cài đặt, điều cần thiết là bạn đã đáp ứng các điều kiện tiên quyết phù hợp với trình quản lý gói của bạn. Chúng tôi khuyên bạn nên sử dụng trình quản lý gói Anaconda vì nó cài đặt tất cả các phụ thuộc.

Bạn vào link <https://pytorch.org/> Chọn nền tảng tương ứng để download và chạy các lệnh như hướng dẫn để cài đặt.



Hình 3.1. Trang tải thư viện Pytorch

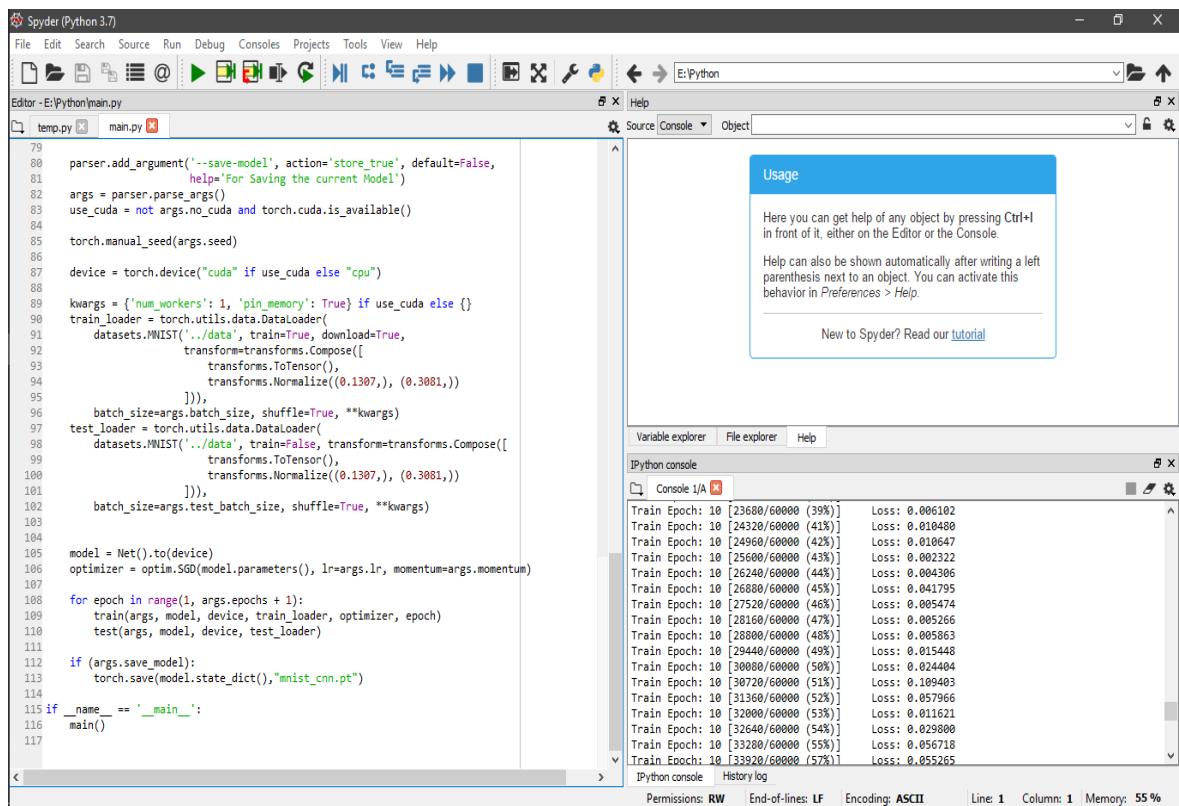
- Bạn copy đoạn trong mục Run thí Command.
- Bạn Khởi động Anaconda Powershell Prompt và Paste đoạn mã vào để tải Pytorch



```
(base) PS C:\Users\vinh-pc> conda install pytorch torchvision cudatoolkit=10.0 -c pytorch
```

Sau đó chờ ít phút để chương trình cài đặt hoàn thành.

- Download bộ ví dụ Pytorch Example và chạy thử ví dụ phân lớp ký tự chữ số viết tay mnist
- + Tải bộ ví dụ Pytorch Example tại: <https://github.com/pytorch/examples>
- + Sau đó chạy chương trình mnist



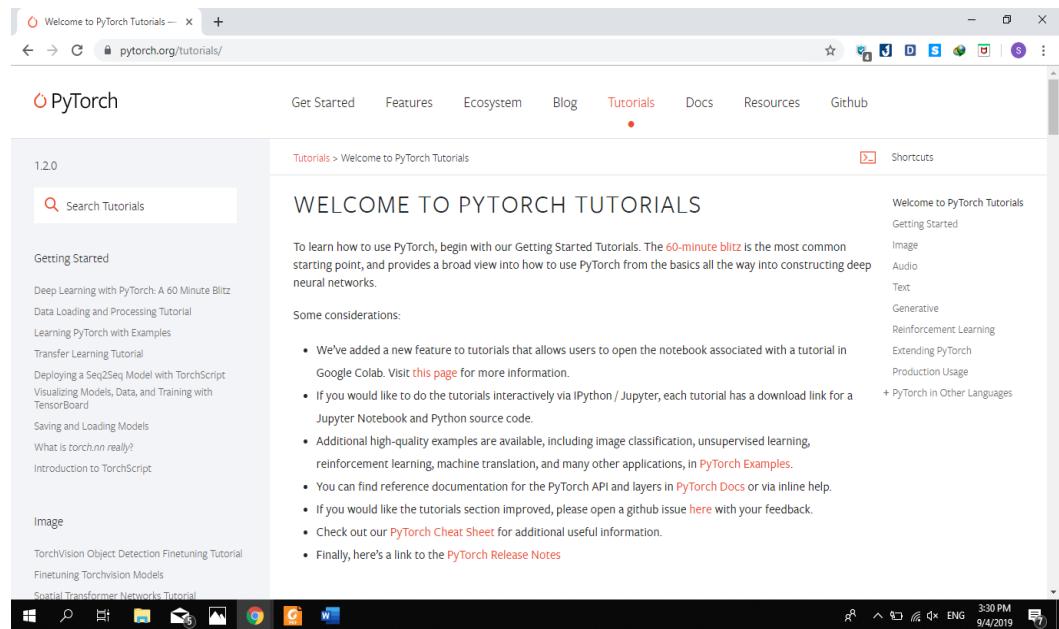
Hình 3.2. Chương trình chữ số viết tay mnist

3.4. Tổng quan về hướng dẫn sử dụng thư viện Pytorch

Hướng dẫn sử dụng thư viện Pytorch:

Truy cập vào trang <https://pytorch.org/tutorials/>

- Trang này trình bày nội dung về các chuyên đề (ở bên trái) được Pytorch hỗ trợ cho người dùng.
- Để tìm hiểu cách sử dụng PyTorch, hãy bắt đầu với Hướng dẫn bắt đầu với Blitz 60 phút, đây là điểm khởi đầu phổ biến nhất và cung cấp một cái nhìn bao quát về cách sử dụng PyTorch từ những điều cơ bản trong suốt quá trình xây dựng mạng Nơron.



Hình 3.3. Trang <https://pytorch.org/tutorials/>

- Một số cân nhắc:

- + Chúng tôi đã thêm một tính năng mới vào hướng dẫn cho phép người dùng mở sổ ghi chép được liên kết với hướng dẫn trong Google Colab. Ghé thăm trang này để biết thêm thông tin.
- + Nếu bạn muốn thực hiện các hướng dẫn tương tác thông qua IPython / Jupyter, mỗi hướng dẫn có một liên kết tải xuống cho Jupyter Notebook và mã nguồn Python.
- + Các ví dụ chất lượng cao bổ sung có sẵn, bao gồm phân loại hình ảnh, học tập không giám sát, học tăng cường, dịch máy và nhiều ứng dụng khác, trong các ví dụ PyTorch.
- + Bạn có thể tìm tài liệu tham khảo cho API PyTorch và các lớp trong PyTorch Docs hoặc thông qua trợ giúp nội tuyến.
- + Nếu bạn muốn phần hướng dẫn được cải thiện, vui lòng mở một vấn đề github ở đây với phản hồi của bạn. Kiểm tra PyTorch Cheat Sheet của chúng tôi để biết thêm thông tin hữu ích.

3.4.1. Mở đầu (Getting Started)

- *Deep Learning Với Pytorch: Một Blitz 60 Phút*

Mục tiêu của hướng dẫn này

- + Hiểu thư viện PyTorch từ Tenor và mạng Nơ-ron ở mức cao.
- + Huấn luyện một mạng lưới thần kinh nhỏ để phân loại hình ảnh
- + Hướng dẫn này giả định rằng bạn có một sự quen thuộc cơ bản của numpy

- *Hướng dẫn tải dữ liệu và tiền xử lý dữ liệu (Data Loading And Processing Tutorial)*

Rất nhiều nỗ lực trong việc giải quyết bất kỳ vấn đề máy học nào đi vào việc chuẩn bị dữ liệu. PyTorch cung cấp nhiều công cụ để tải dữ liệu dễ dàng và hy vọng, để làm cho mã của bạn dễ đọc hơn. Trong hướng dẫn này, chúng ta sẽ xem cách tải và tiền xử lý / dữ liệu tăng thêm từ một bộ dữ liệu không tầm thường.

- *Học Pytorch với ví dụ mẫu(Learning Pytorch With Examples)*

Hướng dẫn này giới thiệu các khái niệm cơ bản của PyTorch thông qua các ví dụ độc lập

Về cốt lõi, PyTorch cung cấp hai tính năng chính:

- + Một Tensor n chiều, tương tự như numpy nhưng có thể chạy trên GPU
- + Tự động phân biệt để xây dựng và đào tạo mạng neural

Chúng tôi sẽ sử dụng mạng ReLU được kết nối đầy đủ làm ví dụ đang chạy. Mạng sẽ có một lớp ẩn duy nhất và sẽ được đào tạo với độ dốc giảm dần để phù hợp với dữ liệu ngẫu nhiên bằng cách giảm thiểu khoảng cách Euclidean giữa đầu ra mạng và đầu ra thực.

- *Hướng dẫn transfer learning (Transfer Learning Tutorial)*

Trong hướng dẫn này, bạn sẽ học cách đào tạo mạng của mình bằng cách học chuyên. Bạn có thể đọc thêm về việc học chuyển tại ghi chú cs231n

Có 2 chuyển giao chính:

- Finetuning the convnet: Thay vì khởi tạo ngẫu nhiên, chúng tôi khởi tạo mạng bằng một mạng được xử lý trước, giống như mạng được đào tạo trên bộ dữ liệu 1000 hình ảnh. Phần còn lại của đào tạo trông như bình thường.
- ConvNet as fixed feature extractor: Ở đây, chúng tôi sẽ đóng băng các trọng số cho tất cả các mạng ngoại trừ lớp cuối cùng được kết nối đầy đủ. Lớp được kết nối đầy đủ cuối cùng này được thay thế bằng một lớp mới với trọng lượng ngẫu nhiên và chỉ có lớp này được đào tạo.

- *Đổi Mô Hình Seq2seq Với Torchscript (Deploying A Seq2seq Model With Torchscript)*

Hướng dẫn này sẽ hướng dẫn quy trình chuyển đổi mô hình tuần tự sang TorchScript bằng API TorchScript. Mô hình mà chúng tôi sẽ chuyển đổi là mô hình chatbot từ hướng dẫn Chatbot. Bạn có thể coi hướng dẫn này là

Hướng dẫn Chatbot 2 và hướng dẫn Chatbot và triển khai mô hình giả định của riêng bạn hoặc bạn có thể bắt đầu với tài liệu này và sử dụng mô hình đã được sàng lọc mà chúng tôi lưu trữ. Trong trường hợp sau, bạn có thể tham khảo hướng dẫn Chatbot ban đầu để biết chi tiết về tiền xử lý dữ liệu, lý thuyết mô hình và định nghĩa và đào tạo mô hình.

- *Mô hình Visualizing Data và đào tạo với Tensorboard (Visualizing Models, Data, And Training With Tensorboard)*

Trong 60 phút Blitz, chúng tôi chỉ cho bạn cách tải dữ liệu, cung cấp thông qua mô hình mà chúng tôi xác định là lớp con của nn.Module, huấn luyện mô hình này về dữ liệu đào tạo và kiểm tra dữ liệu thử nghiệm. Để xem những gì xảy ra, chúng tôi in ra một số thống kê khi mô hình đang đào tạo để hiểu được liệu việc đào tạo có tiến triển hay không. Tuy nhiên, chúng ta có thể làm tốt hơn thế nhiều: PyTorch tích hợp với TensorBoard, một công cụ được thiết kế để trực quan hóa kết quả của các hoạt động đào tạo mạng lưới thần kinh. Hướng dẫn này minh họa một số chức năng của nó, sử dụng bộ dữ liệu Fashion-MNIST có thể được đọc vào PyTorch bằng cách sử dụng Torchvision.datasets.

Trong hướng dẫn này ta học: Đọc dữ liệu và với các biến đổi phù hợp (gần giống với hướng dẫn trước); Thiết lập TensorBoard; Viết thư cho TensorBoard.; Kiểm tra kiến trúc mô hình bằng cách sử dụng TensorBoard; Sử dụng TensorBoard để tạo các phiên bản tương tác của các hình ảnh mà chúng ta đã tạo trong hướng dẫn trước, với ít mã hơn

- *Lưu và tải mô hình (Saving And Loading Models)*

Tài liệu này cung cấp giải pháp cho nhiều trường hợp sử dụng khác nhau liên quan đến việc lưu và tải các mô hình PyTorch. Đừng ngại đọc toàn bộ tài liệu hoặc chỉ cần bỏ qua mã bạn cần cho trường hợp sử dụng mong muốn.

- *Torch.nn thực ra là gì? (What is torch.nn really?)*

Chúng tôi khuyên bạn nên chạy hướng dẫn này dưới dạng sổ ghi chép, không phải là tập lệnh. Để tải xuống tệp sổ ghi chép (.ipynb), nhấp vào liên kết ở đầu trang.

PyTorch cung cấp các mô-đun và các lớp được thiết kế trang nhã Torch.nn, Torch.optim, Dataset và DataLoader để giúp bạn tạo và huấn luyện các mạng thần kinh. Để sử dụng đầy đủ sức mạnh của họ và tùy chỉnh chúng cho vấn đề của bạn, bạn cần thực sự hiểu chính xác những gì họ làm. Để phát triển sự hiểu biết này, trước tiên chúng tôi sẽ đào tạo mạng lưới thần kinh cơ bản trên tập dữ liệu MNIST mà không sử dụng bất kỳ tính năng nào từ các mô hình này; ban đầu chúng tôi sẽ chỉ sử dụng chức năng tenx PyTorch cơ bản

nhất. Sau đó, chúng tôi sẽ tăng dần một tính năng từ Torch.nn, Torch.optim, Dataset hoặc DataLoader tại một thời điểm, hiển thị chính xác từng phần, và cách nó hoạt động để làm cho mã ngắn gọn hơn hoặc linh hoạt hơn.

- *Giới thiệu về Torchscript (Introduction To Torchscript)*

Trong hướng dẫn này, chúng tôi sẽ đề cập đến:

- Những điều cơ bản của tác giả mô hình trong PyTorch, bao gồm:
 - Mô-đun
 - Xác định các hàm chuyển tiếp
 - Kết hợp các mô-đun thành một hệ thống phân cấp của các mô-đun
- Phương thức chuyển đổi các mô-đun PyTorch sang TorchScript, thời gian chạy triển khai hiệu suất cao của chúng tôi
 - Truy tìm một mô-đun hiện có
 - Sử dụng kịch bản để biên dịch trực tiếp một mô-đun
 - Làm thế nào để sáng tác cả hai cách tiếp cận
 - Lưu và tải các mô-đun TorchScript

3.4.2. Ảnh (Image)

- *Hướng dẫn giải quyết đối tượng torchvision*

Đối với hướng dẫn này, chúng ta sẽ hoàn thiện mô hình Mặt nạ R-CNN được đào tạo trước trong Cơ sở dữ liệu của Penn-Fudan để Phát hiện và Phân đoạn Người đi bộ. Nó chứa 170 hình ảnh với 345 trường hợp người đi bộ và chúng tôi sẽ sử dụng nó để minh họa cách sử dụng các tính năng mới trong đèn pin để đào tạo mô hình phân đoạn cá thể trên bộ dữ liệu tùy chỉnh.

- *Mô hình Finetuning torchvision*

Trong hướng dẫn này, chúng ta sẽ xem xét sâu hơn về cách thức tinh chỉnh và tính năng trích xuất các mô hình đèn pin, tất cả đều đã được xử lý trước trên bộ dữ liệu Imagenet 1000 lớp. Hướng dẫn này sẽ cung cấp một cái nhìn sâu sắc về cách làm việc với một số kiến trúc CNN hiện đại và sẽ xây dựng một trực giác để hoàn thiện bất kỳ mô hình PyTorch nào. Vì mỗi kiến trúc mô hình là khác nhau, không có mã hoàn thiện bao tóm tắt nào sẽ hoạt động trong tất cả các kịch bản. Thay vào đó, nhà nghiên cứu phải xem xét kiến trúc hiện có và điều chỉnh tùy chỉnh cho từng mô hình.

Trong tài liệu này, sẽ thực hiện hai loại học chuyển giao: hoàn thiện và trích xuất tính năng. Trong quá trình hoàn thiện, chúng tôi bắt đầu với một mô hình đã được sàng lọc trước và cập nhật tất cả các tham số của mô hình cho nhiệm vụ mới của chúng tôi, về bản chất là đào tạo lại toàn bộ mô hình. Trong trích xuất tính năng, chúng tôi bắt đầu với một mô hình đã được sàng lọc

trước và chỉ cập nhật các trọng số lớp cuối cùng mà chúng tôi rút ra dự đoán. Nó được gọi là trích xuất tính năng vì chúng tôi sử dụng CNN đã xử lý trước như một trình trích xuất tính năng cố định và chỉ thay đổi lớp đầu ra.

- *Hướng dẫn sử dụng mạng chuyển đổi Spatial*

Trong hướng dẫn này, bạn sẽ tìm hiểu cách tăng cường mạng của mình bằng cơ chế chú ý trực quan được gọi là mạng biến áp không gian. Bạn có thể đọc thêm về các mạng biến áp không gian trong bài viết DeepMind

Mạng biến áp không gian là một khái quát của sự chú ý khác nhau đối với bất kỳ chuyển đổi không gian. Mạng biến áp không gian (viết tắt là STN) cho phép mạng thần kinh học cách thực hiện các phép biến đổi không gian trên hình ảnh đầu vào để tăng cường tính bắt biến hình học của mô hình. Ví dụ, nó có thể cắt một vùng quan tâm, chia tỷ lệ và điều chỉnh hướng của hình ảnh. Nó có thể là một cơ chế hữu ích vì các CNN không phải là bắt biến đối với phép quay và tỷ lệ và các phép biến đổi affine tổng quát hơn.

- *Chuyển giao Neural sử dụng Pytorch*

Hướng dẫn này giải thích cách thực hiện thuật toán Kiểu thần kinh được phát triển bởi Leon A. Gatys, Alexander S. Ecker và Matthias Bethge. Neural-Style, hoặc Neural-Transfer, cho phép bạn chụp ảnh và tái tạo nó với một phong cách nghệ thuật mới. Thuật toán lấy ba hình ảnh, hình ảnh đầu vào, hình ảnh nội dung và hình ảnh phong cách và thay đổi đầu vào để giống với nội dung của hình ảnh nội dung và phong cách nghệ thuật của hình ảnh phong cách.

- *Tạo ví dụ nghịch cảnh*

Nếu bạn đang đọc điều này, hy vọng bạn có thể đánh giá cao hiệu quả của một số mô hình học máy. Nghiên cứu không ngừng thúc đẩy các mô hình ML trở nên nhanh hơn, chính xác hơn và hiệu quả hơn. Tuy nhiên, một khía cạnh thường bị bỏ qua của các mô hình thiết kế và đào tạo là bảo mật và sự mạnh mẽ, đặc biệt là khi đối mặt với một kẻ thù muốn đánh lừa mô hình.

Hướng dẫn này sẽ nâng cao nhận thức của bạn về các lỗ hổng bảo mật của các mô hình ML và sẽ cung cấp cái nhìn sâu sắc về chủ đề nóng của học máy đối nghịch. Bạn có thể ngạc nhiên khi thấy rằng việc thêm các nhiễu loạn không thể nhận biết vào một hình ảnh có thể gây ra hiệu suất mô hình khác nhau đáng kể. Cho rằng đây là một hướng dẫn, chúng tôi sẽ khám phá chủ đề thông qua ví dụ trên một bộ phân loại hình ảnh. Cụ thể, chúng tôi sẽ sử dụng một trong những phương thức tấn công đầu tiên và phổ biến nhất, Fast Gradient Sign Attack (FGSM), để đánh lừa trình phân loại MNIST.

3.4.3. Âm thanh (Audio)

- *Hướng dẫn Torchaudio*

PyTorch là một nền tảng học tập sâu nguồn mở cung cấp một bộ trình liên mạch từ tạo mẫu nghiên cứu đến triển khai sản xuất với sự hỗ trợ GPU.

Nỗ lực đáng kể trong việc giải quyết các vấn đề máy học đi vào việc chuẩn bị dữ liệu. Torchaudio tận dụng hỗ trợ GPU PyTorch, và cung cấp nhiều công cụ để tải dữ liệu dễ dàng và dễ đọc hơn. Trong hướng dẫn này, chúng ta sẽ xem cách tải và tiền xử lý dữ liệu từ một tập dữ liệu đơn giản.

3.4.4. Văn bản (Text)

- *Hướng dẫn Chatbox*

Trong hướng dẫn này, khám phá một trường hợp sử dụng thú vị và thú vị của các mô hình tuần tự lặp lại liên tục. Chúng tôi sẽ đào tạo một chatbot đơn giản bằng cách sử dụng các kịch bản phim từ Cornell Movie-Dialogs Corpus.

Mô hình đàm thoại là một chủ đề nóng trong nghiên cứu trí tuệ nhân tạo. Chatbots có thể được tìm thấy trong một loạt các cài đặt, bao gồm các ứng dụng dịch vụ khách hàng và bộ phận trợ giúp trực tuyến. Các bot này thường được cung cấp bởi các mô hình dựa trên truy xuất, đưa ra các câu trả lời được xác định trước cho các câu hỏi của các hình thức nhất định. Trong một miền bị hạn chế cao như bộ phận trợ giúp CNTT của công ty, các mô hình này có thể là đủ, tuy nhiên, chúng không đủ mạnh cho các trường hợp sử dụng chung hơn. Dạy một cỗ máy để thực hiện một cuộc trò chuyện có ý nghĩa với một con người trong nhiều lĩnh vực là một câu hỏi nghiên cứu còn lâu mới được giải quyết. Gần đây, sự bùng nổ học tập sâu đã cho phép các mô hình tạo thế hệ mạnh mẽ như Mô hình đàm thoại thần kinh của Google, đánh dấu một bước tiến lớn đối với các mô hình đàm thoại thế hệ đa miền. Trong hướng dẫn này, chúng tôi sẽ triển khai loại mô hình này trong PyTorch.

- *Phân loại tên với đặc điểm RNN*

Chúng tôi sẽ xây dựng và đào tạo một RNN cấp độ nhân vật cơ bản để phân loại các từ. Một RNN cấp độ ký tự đọc các từ dưới dạng một loạt các ký tự - xuất ra một dự đoán và trạng thái ẩn ẩn ở mỗi bước, đưa trạng thái ẩn trước đó vào từng bước tiếp theo. Chúng tôi lấy dự đoán cuối cùng là đầu ra, tức là từ đó thuộc về lớp nào.

- *Tạo tên với đặc điểm RNN*

Trong hướng dẫn này chúng tôi sẽ tạo tên từ các ngôn ngữ.

- *Deep Learning Cho NLP Với Pytorch*

Hướng dẫn này sẽ hướng dẫn bạn qua các ý tưởng chính của lập trình học sâu bằng Pytorch. Nhiều trong số các khái niệm (chẳng hạn như trừu tượng hóa biểu đồ tính toán và tự động) không phải là duy nhất đối với Pytorch và có liên quan đến bất kỳ bộ công cụ học tập sâu nào ngoài kia.

Hướng dẫn này để tập trung chủ đề vào NLP cho những người chưa bao giờ viết mã trong bất kỳ khuôn khổ học tập sâu nào (ví dụ: TensorFlow, Theano, Keras, Dynet). Nó giả định kiến thức làm việc về các vấn đề NLP cốt lõi: gắn thẻ một phần lời nói, mô hình hóa ngôn ngữ, v.v. Nó cũng giả định sự quen thuộc với các mạng thần kinh ở cấp độ của một lớp AI giới thiệu (chẳng hạn như từ cuốn sách của Russel và Norvig). Thông thường, các khóa học này bao gồm thuật toán backpropagation cơ bản trên các mạng thần kinh chuyển tiếp thức ăn, và đưa ra quan điểm rằng chúng là các chuỗi các thành phần của tuyến tính và phi tuyến tính. Hướng dẫn này nhằm mục đích giúp bạn bắt đầu viết mã học sâu, với điều kiện bạn có kiến thức tiên quyết này.

Lưu ý đây là về mô hình, không phải dữ liệu. Đối với tất cả các mô hình, tôi chỉ tạo một vài ví dụ thử nghiệm với kích thước nhỏ để bạn có thể thấy trọng lượng thay đổi như thế nào khi nó luyện tập. Nếu bạn có một số dữ liệu thực sự muốn thử, bạn sẽ có thể trích xuất bất kỳ mô hình nào từ sổ ghi chép này và sử dụng chúng trên đó.

- *Chuyển đổi với mạng Sequence To Sequence và Attention*

Trong dự án này, sẽ dạy một mạng neural để dịch từ tiếng Pháp sang tiếng Anh.

- *Hướng dẫn phân loại văn bản*

Hướng dẫn này chỉ ra cách sử dụng bộ dữ liệu phân loại văn bản

3.4.5. Generative

- *Hướng dẫn DCGAN*

Hướng dẫn này sẽ giới thiệu về DCGAN thông qua một ví dụ. Chúng tôi sẽ đào tạo một mạng lưới đối thủ thế hệ (GAN) để tạo ra những người nổi tiếng mới sau khi cho nó xem hình ảnh của nhiều người nổi tiếng thực sự. Hầu hết các mã ở đây là từ việc triển khai dcgan trong pytorch / ví dụ, và tài liệu này sẽ đưa ra lời giải thích kỹ lưỡng về việc triển khai và làm sáng tỏ về cách thức và lý do mô hình này hoạt động. Nhưng đừng lo lắng, không cần có kiến thức trước về GAN, nhưng nó có thể cần một người hẹn giờ đầu tiên để dành thời gian suy luận về những gì thực sự xảy ra dưới mui xe. Ngoài ra, vì lợi ích của thời gian, nó sẽ giúp có một hoặc hai GPU. Hãy bắt đầu từ đầu.

3.4.6. Học tăng cường (Reinforcement Learning)

- *Hướng dẫn Reinforcement Learning (DQN)*

Hướng dẫn này chỉ ra cách sử dụng PyTorch để huấn luyện một tác nhân Deep Q Learning (DQN) trong nhiệm vụ CartPole-v0 từ OpenAI Gym

3.4.7. Mở rộng Pytorch

- *Tạo khai thác sử dụng Numpy và Scipy*

Trong hướng dẫn này, chúng ta sẽ trải qua hai nhiệm vụ:

- Tạo một lớp mạng thần kinh không có tham số: Điều này gọi vào numpy như là một phần của việc thực hiện
- Tạo một lớp mạng thần kinh có trọng lượng có thể học được: Điều này gọi vào SciPy như là một phần của việc thực hiện

- *Tùy chỉnh C++ và khai thác Cuda*

PyTorch cung cấp rất nhiều hoạt động liên quan đến mạng lưới thần kinh, đại số tensor tùy ý, sắp xếp dữ liệu và các mục đích khác. Tuy nhiên, bạn vẫn có thể thấy mình cần một hoạt động tùy biến hơn. Ví dụ: bạn có thể muốn sử dụng chức năng kích hoạt mới mà bạn tìm thấy trong một bài báo hoặc thực hiện một thao tác bạn đã phát triển như một phần của nghiên cứu.

Cách dễ nhất để tích hợp một thao tác tùy chỉnh như vậy trong PyTorch là viết nó bằng Python bằng cách mở rộng **Function** và **module** như được nêu ở đây. Điều này cung cấp cho bạn toàn bộ sức mạnh của sự khác biệt tự động (không cho bạn viết các hàm phái sinh) cũng như tính biểu cảm thông thường của Python. Tuy nhiên, có thể đôi khi hoạt động của bạn được thực hiện tốt hơn trong C++. Ví dụ, mã của bạn có thể cần phải thực sự nhanh vì nó được gọi rất thường xuyên trong mô hình của bạn hoặc rất tốn kém ngay cả đối với một vài cuộc gọi. Một lý do hợp lý khác là nó phụ thuộc hoặc tương tác với các thư viện C hoặc C++ khác. Để giải quyết các trường hợp như vậy, PyTorch cung cấp một cách rất dễ dàng để viết các phần mở rộng C++ tùy chỉnh.

Các tiện ích mở rộng C++ là một cơ chế chúng tôi đã phát triển để cho phép người dùng (bạn) tạo các toán tử PyTorch được xác định ngoài nguồn, tức là tách biệt với phụ trợ PyTorch. Cách tiếp cận này khác với cách thức hoạt động của PyTorch bản địa. Các tiện ích mở rộng C++ nhằm dành cho bạn phần lớn bắn tóm tắt liên quan đến việc tích hợp một hoạt động với phụ trợ PyTorch, đồng thời cung cấp cho bạn mức độ linh hoạt cao cho các dự án dựa trên PyTorch của bạn. Tuy nhiên, một khi bạn đã xác định hoạt động của mình là một phần mở rộng C++, biến nó thành một hàm PyTorch riêng phần

lớn là vấn đề của tổ chức mã, bạn có thể giải quyết sau thực tế nếu bạn quyết định đóng góp hoạt động của mình ngược dòng.

- *Khai thác TorchCript với hoạt động C++*

Bản phát hành PyTorch 1.0 đã giới thiệu một mô hình lập trình mới cho PyTorch có tên TorchScript. TorchScript là tập hợp con của ngôn ngữ lập trình Python có thể được phân tích cú pháp, biên dịch và tối ưu hóa bởi trình biên dịch TorchScript. Hơn nữa, các mô hình TorchScript được biên dịch có tùy chọn được tuân tự hóa thành định dạng tệp trên đĩa, sau đó bạn có thể tải và chạy từ C ++ thuần túy (cũng như Python) để suy luận.

TorchScript hỗ trợ một tập hợp lớn các hoạt động được cung cấp bởi gói **torch**, cho phép bạn thể hiện nhiều loại mô hình phức tạp hoàn toàn như một chuỗi các hoạt động kéo căng từ thư viện tiêu chuẩn PyTorch xông vào. Tuy nhiên, có thể đôi khi bạn thấy mình cần mở rộng TorchScript bằng chức năng C ++ hoặc CUDA tùy chỉnh. Mặc dù chúng tôi khuyên bạn chỉ nên sử dụng tùy chọn này nếu ý tưởng của bạn không thể được thể hiện (đủ hiệu quả) như một hàm Python đơn giản, chúng tôi cung cấp giao diện rất thân thiện và đơn giản để xác định các hạt nhân C ++ và CUDA tùy chỉnh bằng cách sử dụng thang đo C ++ hiệu suất cao của PyTorch thư viện. Khi đã gắn kết với TorchScript, bạn có thể nhúng các hạt nhân tùy chỉnh này (hoặc các op ops) vào mô hình TorchScript của bạn và thực hiện cả hai trong Python và ở dạng tuân tự của chúng trực tiếp trong C ++

Các đoạn sau đây đưa ra một ví dụ về việc viết **op** tùy chỉnh TorchScript để gọi vào OpenCV, một thư viện thị giác máy tính được viết bằng C ++. Chúng tôi sẽ thảo luận về cách làm việc với các tenxơ trong C ++, cách chuyển đổi chúng thành các định dạng tenor của bên thứ ba một cách hiệu quả (trong trường hợp này là OpenCV ` ` Mat` `), cách đăng ký toán tử của bạn với thời gian chạy TorchScript và cuối cùng là cách biên dịch toán tử và sử dụng nó trong Python và C ++.

3.4.8. Sử dụng và sản xuất

- *Mô hình thực tập song song tốt nhất*

Dữ liệu song song và mô hình song song được sử dụng rộng rãi trong các kỹ thuật đào tạo phân tán. Các bài đăng trước đây đã giải thích cách sử dụng DataParallel để huấn luyện một mạng thần kinh trên nhiều GPU. DataParallel sao chép cùng một mô hình cho tất cả các GPU, trong đó mỗi GPU tiêu thụ một phân vùng khác nhau của dữ liệu đầu vào. Mặc dù nó có thể tăng tốc đáng kể quá trình đào tạo, nhưng nó không hoạt động đối với một số trường hợp sử dụng khi mô hình quá lớn để phù hợp với một GPU duy nhất. Bài đăng này cho thấy cách giải quyết vấn đề đó bằng cách sử dụng mô hình

song song và cũng chia sẻ một số hiểu biết về cách tăng tốc đào tạo song song mô hình.

Ý tưởng cấp cao của mô hình song song là đặt các mạng con khác nhau của mô hình lên các thiết bị khác nhau và thực hiện phương thức chuyển tiếp phù hợp để di chuyển đầu ra trung gian trên các thiết bị. Vì chỉ là một phần của mô hình hoạt động trên mọi thiết bị riêng lẻ, một bộ thiết bị có thể phục vụ chung cho một mô hình lớn hơn. Trong bài đăng này, chúng tôi sẽ không cố gắng xây dựng các mô hình khổng lồ và ép chúng vào một số lượng GPU hạn chế. Thay vào đó, bài đăng này tập trung vào việc hiển thị ý tưởng của mô hình song song. Người đọc có thể áp dụng các ý tưởng cho các ứng dụng trong thế giới thực.

- *Bắt đầu với phân phối song song*

DistributDataParallel (DDP) thực hiện song song dữ liệu ở cấp độ mô-đun. Nó sử dụng các tập thể truyền thông trong gói Torch.distribution để đồng bộ hóa độ dốc, tham số và bộ đệm. Song song có sẵn cả trong một quy trình và trên các quy trình. Trong một quy trình, DDP sao chép mô-đun đầu vào thành các thiết bị được chỉ định trong **device_ids**, phân tán các đầu vào đọc theo kích thước lô tương ứng và tập hợp các đầu ra cho **output_device**, tương tự như DataParallel. Trên các quy trình, DDP chèn đồng bộ hóa tham số cần thiết vào các lần chuyển tiếp và đồng bộ hóa độ dốc trong các lần truyền ngược. Người dùng tùy thuộc vào việc ánh xạ các quy trình tới các tài nguyên có sẵn, miễn là các quy trình không chia sẻ các thiết bị GPU. Cách tiếp cận được đề xuất (thường là nhanh nhất) là tạo một quy trình cho mọi bản sao mô-đun, tức là không có bản sao mô-đun trong một quy trình. Mã trong hướng dẫn này chạy trên máy chủ 8-GPU, nhưng nó có thể dễ dàng khai thác cho các môi trường khác.

- *Viết ứng dụng phân phối với pytorch*

Trong hướng dẫn ngắn này, chúng ta sẽ tìm hiểu về gói phân phối của PyTorch. Chúng tôi sẽ xem cách thiết lập cài đặt phân tán, sử dụng các chiến lược truyền thông khác nhau và xem qua một số nội bộ của gói.

- *Nâng cấp Pytorch và Xây dựng REST API sử dụng flask*

Trong hướng dẫn này, chúng tôi sẽ triển khai mô hình PyTorch bằng Flask và hiển thị API REST cho suy luận mô hình. Cụ thể, chúng tôi sẽ triển khai mô hình DenseNet 121 đã được phát hiện trước để phát hiện hình ảnh.

- *Pytorch 1.0 đào tạo phân phối với Amazon AWS*

Trong hướng dẫn này, trình bày cách thiết lập, mã hóa và chạy trình đào tạo phân tán PyTorch 1.0 trên hai nút Amazon AWS đa gpu. Chúng ta sẽ

bắt đầu với việc mô tả thiết lập AWS, sau đó là cấu hình môi trường PyTorch và cuối cùng là mã cho trình huấn luyện phân tán. Hy vọng rằng bạn sẽ thấy rằng thực sự có rất ít thay đổi mã cần thiết để mở rộng mã đào tạo hiện tại của bạn sang một ứng dụng phân tán và hầu hết các công việc đều nằm trong thiết lập môi trường một lần.

- *Tải mô hình Pytorch trong C++*

Như tên gọi của nó, giao diện chính cho PyTorch là ngôn ngữ lập trình Python. Mặc dù Python là ngôn ngữ phù hợp và được ưa thích cho nhiều tình huống đòi hỏi tính năng động và dễ lặp lại, nhưng cũng có nhiều tình huống trong đó chính xác các thuộc tính này của Python là không thuận lợi. Một môi trường mà sau này thường áp dụng là sản xuất - vùng đất có độ trễ thấp và yêu cầu triển khai nghiêm ngặt. Đối với các kịch bản sản xuất, C ++ thường là ngôn ngữ được lựa chọn, ngay cả khi chỉ liên kết nó với ngôn ngữ khác như Java, Rust hoặc Go. Các đoạn sau đây sẽ phác thảo đường dẫn PyTorch cung cấp để đi từ mô hình Python hiện có sang biểu diễn tuần tự có thể được tải và thực hiện hoàn toàn từ C ++, không phụ thuộc vào Python.

- *Xuất Khẩu Mô Hình Từ Pytorch Đến Onnx Và Chạy Nó Sử Dụng Onnx Runtime*

Trong hướng dẫn này, chúng tôi mô tả cách chuyển đổi một mô hình được xác định trong PyTorch sang định dạng ONNX và sau đó chạy nó với

ONNX Runtime. ONNX Runtime là công cụ tập trung vào hiệu năng cho các mô hình ONNX, suy luận hiệu quả trên nhiều nền tảng và phần cứng (Windows, Linux và Mac và trên cả CPU và GPU). ONNX Runtime đã chứng minh tăng hiệu suất đáng kể trên nhiều mô hình.

Đối với hướng dẫn này, bạn sẽ cần cài đặt ONNX và ONNX Runtime. Bạn có thể nhận các bản dựng nhị phân của ONNX và ONNX Runtime với cài đặt pip onnx onnxr nb. Lưu ý rằng ONNX Runtime tương thích với các phiên bản Python 3.5 đến 3.7.

3.4.9. Pytorch trong các ngôn ngữ khác

- *Sử dụng pytorch C++ Frontend*

Giao diện PyTorch C ++ là giao diện C ++ thuận tiện cho khung máy học PyTorch. Mặc dù giao diện chính của PyTorch tự nhiên là Python, API Python này nằm trên một cơ sở mã C ++ đáng kể cung cấp các cấu trúc dữ liệu cơ bản và chức năng như tenor và phân biệt tự động. Giao diện C ++ hiển thị API C ++ 11 thuận tiện mở rộng cơ sở mã C ++ cơ bản này với các công cụ cần thiết cho đào tạo và suy luận học máy. Điều này bao gồm một bộ sưu tập tích hợp các thành phần phổ biến cho mô hình mạng thần kinh; một API

để mở rộng bộ sưu tập này với các mô-đun tùy chỉnh; một thư viện các thuật toán tối ưu hóa phổ biến như giảm dần độ dốc ngẫu nhiên; trình tải dữ liệu song song với API để xác định và tải bộ dữ liệu; thói quen tuân tự hóa và nhiều hơn nữa.

Hướng dẫn này sẽ hướng dẫn bạn qua một ví dụ từ đầu đến cuối về việc đào tạo một mô hình với giao diện C++. Cụ thể, chúng tôi sẽ đào tạo một DCGAN - một loại mô hình thế hệ - để tạo ra hình ảnh của các chữ số MNIST. Mặc dù về mặt khái niệm là một ví dụ đơn giản, nó cũng đủ để cung cấp cho bạn một cái nhìn tổng quan về cách hoạt động của frontend PyTorch C++ và làm giảm sự thèm ăn của bạn để đào tạo các mô hình phức tạp hơn. Chúng tôi sẽ bắt đầu với một số từ thúc đẩy lý do tại sao bạn muốn sử dụng giao diện C++ để bắt đầu, sau đó đi thẳng vào việc xác định và đào tạo mô hình của chúng tôi.

3.5. Deep Learning Với Pytorch: Một Blitz 60 Phút

3.5.1. Pytorch là gì?

❖ Tensor

Tensors tương tự gần giống như mảng NumPy's ndarrays với sự bổ sung thêm là Tensors có thể được sử dụng trên một GPU để tăng tốc máy tính.

 Chú thích:

Một ma trận chưa được khởi tạo được khai báo, nhưng không chứa các giá trị xác định đã biết trước khi nó được sử dụng. Khi một ma trận chưa được khởi tạo được tạo, bất kỳ giá trị nào trong bộ nhớ được phân bổ tại thời điểm đó sẽ xuất hiện dưới dạng các giá trị ban đầu

Chương trình mẫu:

- ✓ **Tập tin:** tensor_tutorial.py
- ✓ **Nội dung:**

#Gọi thư viện

```
from __future__ import print_function  
import torch
```

Hàm tương đương với hàm main trong C/C++

```
def main():
```

Xây dựng ma trận 5x3, chưa được khởi tạo:

```
x = torch.empty(5, 3)  
print(x)
```

Xuất kết quả:

```
tensor([[8.4490e-39, 1.0469e-38, 9.3674e-39],
```

```
[9.9184e-39, 8.7245e-39, 9.2755e-39],  
[8.9082e-39, 9.9184e-39, 8.4490e-39],  
[9.6429e-39, 1.0653e-38, 1.0469e-38],  
[4.2246e-39, 1.0378e-38, 9.6429e-39]])
```

Xây dựng một ma trận với các phần tử ngẫu nhiên:
x = torch.rand(5, 3)

Xuất kết quả:

```
tensor([[0.9122, 0.4500, 0.2284],  
       [0.4735, 0.4220, 0.1565],  
       [0.1914, 0.7626, 0.0551],  
       [0.9377, 0.6026, 0.9883],  
       [0.2482, 0.2032, 0.2157]])
```

Xây dựng ma trận rỗng (phần tử là số 0) và kiểu dữ liệu dtype
của các phần tử là kiểu long:

```
x = torch.zeros(5, 3, dtype=torch.long)  
print(x)
```

Xuất kết quả:

```
tensor([[0, 0, 0],  
       [0, 0, 0],  
       [0, 0, 0],  
       [0, 0, 0],  
       [0, 0, 0]])
```

#Xây dựng một tensor trực tiếp từ dữ liệu đã có:
x = torch.tensor([5.5, 3])
print(x)

Xuất kết quả:

```
tensor([5.5000, 3.0000])
```

Hoặc tạo ra một tensor dựa trên một tensor đã có. Các phương thức methods này sẽ tái sử dụng các thuộc tính của tensor được tạo đã có trước đó, ví dụ: kiểu dữ liệu dtype nếu trừ khi giá trị mới được cung cấp bởi người dùng.

```
x = x.new_ones(5, 3, dtype=torch.double)      # phương thức  
new_* methods dựa trên kích thước đã cho  
print(x)  
x = torch.randn_like(x, dtype=torch.float)      # ghi đè nạp  
chồng lên dtype!  
print(x)                                      # Kết quả có cùng kích thước
```

Xuất kết quả:

```
tensor([[1., 1., 1.],  
       [1., 1., 1.],  
       [1., 1., 1.],  
       [1., 1., 1.],  
       [1., 1., 1.]], dtype=torch.float64)  
tensor([-1.7608, 2.3644, -0.4953],  
      [ 1.3313, -1.3619, -0.9252],  
      [-0.1510, -1.0781, -0.4981],  
      [ 0.3445, 0.7645, -0.1500],  
      [ 0.7956, 0.0494, -0.8140]])
```

```
# Lấy kích thước  
print(x.size())
```

Xuất kết quả:

```
torch.Size([5, 3])
```

Lưu ý: torch.Size thực tế là một bộ tuple, vì vậy nó hỗ trợ tất cả các hoạt động của bộ dữ liệu.

❖ *Các hoạt động phép toán*

Có nhiều cú pháp cho hoạt động phép toán. Trong ví dụ sau, chúng ta sẽ xem về các hoạt động phép toán cộng.

```
# Addition: cú pháp 1  
y = torch.rand(5, 3)  
print(x + y)
```

Xuất kết quả:

```
tensor([[ 0.7707,  1.1077,  0.2789],  
       [-0.4228,  2.3807,  1.9151],  
       [-0.9157,  0.6138, -0.3533],  
       [-1.3772,  0.3117,  0.8766],  
       [ 0.2261,  0.5641, -0.7411]])
```

```
# Addition: cú pháp 2  
print(torch.add(x, y))
```

Xuất kết quả:

```
tensor([[ 0.7707,  1.1077,  0.2789],  
       [-0.4228,  2.3807,  1.9151],  
       [-0.9157,  0.6138, -0.3533],
```

```
[-1.3772, 0.3117, 0.8766],  
[ 0.2261, 0.5641, -0.7411]])
```

```
# Addition: Cung cấp một tensor đầu ra output làm đối số  
result = torch.empty(5, 3)  
torch.add(x, y, out=result)  
print(result)
```

Xuất kết quả:

```
tensor([[ 0.7707,  1.1077,  0.2789],  
       [-0.4228,  2.3807,  1.9151],  
       [-0.9157,  0.6138, -0.3533],  
       [-1.3772,  0.3117,  0.8766],  
       [ 0.2261,  0.5641, -0.7411]])
```

```
# Addition: in-place  
# adds x to y  
y.add_(x)  
print(y)
```

Xuất kết quả:

```
tensor([[ 0.7707,  1.1077,  0.2789],  
       [-0.4228,  2.3807,  1.9151],  
       [-0.9157,  0.6138, -0.3533],  
       [-1.3772,  0.3117,  0.8766],  
       [ 0.2261,  0.5641, -0.7411]])
```

Tất cả các phương thức method trên đều cho ra xuất kết quả giống nhau.

Lưu ý: Bất kỳ hoạt động nào làm biến đổi một tensor tại chỗ đều được sửa sau với một dấu _ . VD: x.copy_(y), x.t_(), sẽ thay đổi x

#Bạn có thể sử dụng chỉ số index giống như NumPy

```
print(x[:, 1])
```

Xuất kết quả:

```
tensor([ 0.4588,  1.4546, -0.2412, -0.3609, -0.0513])
```

Thay đổi kích thước: Nếu bạn muốn resize/reshape tensor, bạn có thể dùng torch.view

```
x = torch.randn(4, 4)  
y = x.view(16)  
z = x.view(-1, 8) # Kích thước -1 được suy ra từ các số nguyên  
khác
```

```
print(x.size(), y.size(), z.size())
```

Xuất kết quả:

```
torch.Size([4, 4]) torch.Size([16]) torch.Size([2, 8])
```

```
#Nếu bạn có một phần tử tensor, sử dụng .item() để lấy giá trị dưới dạng số Python:  
x = torch.randn(1)  
print(x)  
print(x.item())
```

Xuất kết quả:

```
tensor([0.3864])  
0.38644298911094666
```

❖ Numpy Bridge

Chuyển đổi một Torch Tensor thành một mảng NumPy và ngược lại là một cách dễ dàng (breeze).

Các Torch Tensor và mảng NumPy sẽ chia sẻ các vị trí bộ nhớ của chúng và thay đổi một vị trí này sẽ thay đổi vị trí kia.

✚ Chuyển đổi một Torch Tensor thành một mảng Numpy

```
a = torch.ones(5)  
print(a)
```

Xuất kết quả:

```
tensor([1., 1., 1., 1., 1.])  
  
b = a.numpy()  
  
print(b)
```

Xuất kết quả:

```
[1. 1. 1. 1. 1.]
```

```
#Xem cách mảng numpy thay đổi về giá trị.  
a.add_(1)  
print(a)  
print(b)
```

Xuất kết quả:

```
tensor([2., 2., 2., 2., 2.])
[2. 2. 2. 2. 2.]
```

➡ Chuyển đổi mảng Numpy thành Torch Tensor

```
#Xem cách thay đổi mảng np đã thay đổi Torch Tenor tự động
import numpy as np
a = np.ones(5)
b = torch.from_numpy(a)
np.add(a, 1, out=a)
print(a)
print(b)
```

Xuất kết quả :

```
[2. 2. 2. 2. 2.]
tensor([2., 2., 2., 2., 2.], dtype=torch.float64)
```

Tất cả các Tensors trên CPU ngoại trừ hỗ trợ CharTensor chuyển thành NumPy và ngược lại.

➡ Cuda Tensors

Tensors có thể được chuyển vào bất kỳ thiết bị bằng cách sử dụng phương thức method .to

```
# Hãy chạy ô này chỉ khi có sẵn CUDA
# Chúng ta sẽ sử dụng các đối tượng ``torch.device`` để di
# chuyển các tensors vào và ra khỏi GPU
if torch.cuda.is_available():
    device = torch.device("cuda")      # đổi tượng thiết bị CUDA
    y = torch.ones_like(x, device=device) # Trực tiếp tạo ra một
    # tensor trên GPU
    x = x.to(device)                  # hoặc chỉ sử dụng chuỗi ``.to
    ("cuda") ``

    z = x + y
    print(z)
    print(z.to("cpu", torch.double))   # ``.to`` cũng có thể thay
    # đổi dtype với nhau!
```

Xuất kết quả :

```
tensor([2.9218], device='cuda:0')
tensor([2.9218], dtype=torch.float64)
```

3.5.2. Autograd: Automatic Differentiation (Autograd: Tính đạo hàm tự động)

Trung tâm của tất cả các mạng neural trong PyTorch là autograd gói. Trước tiên chúng ta hãy truy cập ngắn gọn về điều này và sau đó chúng tôi sẽ đi đào tạo mạng neural đầu tiên của chúng tôi.

Các autograd gói cung cấp phân tự động cho tất cả các hoạt động trên tensors. Đó là một khung xác định do chạy, có nghĩa là backprop của bạn được xác định bởi cách mã của bạn được chạy và mỗi lần lặp có thể khác nhau.

Hãy cho chúng tôi thấy điều này trong các điều khoản đơn giản hơn với một số ví dụ.

Tensor

`torch.Tensor` là lớp trung tâm của gói. Nếu bạn đặt thuộc tính của nó `.requires_grad` là `True`, nó bắt đầu theo dõi tất cả các hoạt động trên nó. Khi bạn hoàn thành tính toán của mình, bạn có thể gọi `.backward()` và có tất cả các gradient được tính toán tự động. Độ dốc cho tensor này sẽ được tích lũy vào `.grad` thuộc tính.

Để ngăn một tensor theo dõi lịch sử, bạn có thể gọi `.detach()` để tách nó khỏi lịch sử tính toán và để ngăn chặn tính toán trong tương lai khỏi bị theo dõi.

Để ngăn lịch sử theo dõi (và sử dụng bộ nhớ), bạn cũng có thể bọc khói mã. Điều này có thể đặc biệt hữu ích khi đánh giá một mô hình vì mô hình có thể có các tham số có thể huấn luyện được, nhưng chúng ta không cần độ dốc. `with torch.no_grad(): requires_grad=True`

Có thêm một lớp học rất quan trọng đối với việc thực hiện autograd - a Function.

Tensor và Function được kết nối với nhau và xây dựng một biểu đồ tuần hoàn, mã hóa toàn bộ lịch sử tính toán. Mỗi tensor có một `.grad_fn` thuộc tính tham chiếu đến một thuộc tính Function đã tạo ra Tensor (ngoại trừ các Tensor được tạo bởi người dùng - của chúng). `.grad_fn` is `None`

Nếu bạn muốn tính toán các đạo hàm, bạn có thể gọi `.backward()` trên một Tensor. Nếu Tensor là một vô hướng (nghĩa là nó chứa dữ liệu một phần tử), bạn không cần chỉ định bất kỳ đối số nào `backward()`, tuy nhiên nếu nó có nhiều phần tử hơn, bạn cần chỉ định một gradient đối số là một thang đo có hình dạng phù hợp.

Chương trình mẫu:

- ✓ **Tập tin:** autograd_tutorial.py
- ✓ **Nội dung:**

```
# gọi thư viện
import torch
def main():
    # Tạo một tensor và thiết lập requires_grad=True để theo dõi tính toán với
    # nó
    x = torch.ones(2, 2, requires_grad=True)
    print(x)

    # Thực hiện thao tác kéo căng:
    y = x + 2
    print(y)

    # ``y`` được tạo ra như là kết quả của một hoạt động, vì vậy nó có một
    # ``grad_fn``.
    print(y.grad_fn)

    # Thực hiện nhiều thao tác hơn trên ``y``
    z = y * y * 3
    out = z.mean()

    print(z, out)

    # ``.requires_grad_ (...)`` thay đổi một `Tenor` hiện có của `Yêu
    # cầu_grad`
    # cờ tại chỗ. Cờ đầu vào mặc định là ``Sai`` nếu không được đưa ra.
    a = torch.randn(2, 2)
    a = ((a * 3) / (a - 1))
    print(a.requires_grad)
    a.requires_grad_(True)
    print(a.requires_grad)
    b = (a * a).sum()
    print(b.grad_fn)
```

Gradients

```
# Gradients
# -----
# Hãy backprop ngay bây giờ.
# Bởi vì ``out`` chứa một vô hướng, ``out.backward()`` là
# tương đương với ``out.backward(Torch.tensor(1.))``.
out.backward()

# In gradient d (ra) / dx
print(x.grad)
```

```

# Bây giờ chúng ta hãy xem xét một ví dụ về sản phẩm vector-Jacobian:
x = torch.randn(3, requires_grad=True)
y = x * 2
while y.data.norm() < 1000:
    y = y * 2
print(y)

# Bây giờ trong trường hợp này 'y' không còn là vô hướng. '' Torch.
# autograd ''
# không thể tính toán đầy đủ của Jacobian trực tiếp, nhưng nếu chúng ta
# chỉ
# muốn sản phẩm vector-Jacobian, chỉ cần vượt qua vector để
# ''backward'' như đối số:
v = torch.tensor([0.1, 1.0, 0.0001], dtype=torch.float)
y.backward(v)
print(x.grad)

# Bạn cũng có thể dùng autograd từ lịch sử theo dõi trên Tensors
# với ''.requires_grad = True '' bằng cách gói khối mã trong
# ''với ngọn đuốc. No _ Grad (): ''
print(x.requires_grad)
print((x ** 2).requires_grad)
with torch.no_grad():
    print((x ** 2).requires_grad)

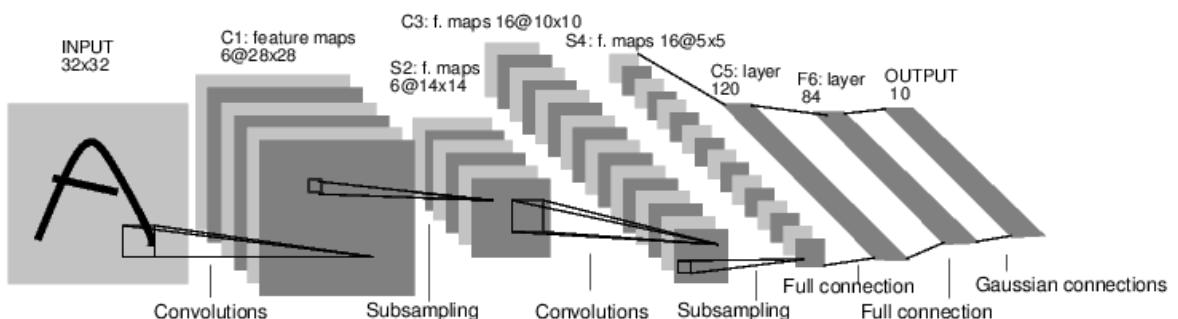
# Gọi thực hiện hàm main
if __name__ == "__main__":
    main()

```

3.5.3. Mạng noron

Mạng noron có thể được xây dựng bằng cách sử dụng gói *package torch.nn autograd*, nn thuộc *autograd* để xác định các mô hình và phân biệt chúng. Một *nn.Module* chứa các lớp và một *method forward(input)* trả về *output*

Ví dụ, hãy xem mạng này phân loại các ảnh số:



Hình 3.4. Mạng này phân loại các ảnh số

convnet

Đây là một mạng chuyển tiếp đơn giản. Nó truyền qua một số tám lõc này sang tám lõc khác và cuối cùng cho đầu ra.

Một quy trình đào tạo điển hình cho mạng nơron như sau:

- Xác định mạng nơron có một số thông số có thể học (hoặc trọng số weights)
- Lặp lại các tập dữ liệu đầu vào
- Xử lý đầu vào thông qua mạng nơron
- Tính toán (khoảng cách từ đầu ra đến kết quả)
- Lan truyền các gradients trở lại các tham số của mạng nơron
- Cập nhật trọng số weights của mạng, thường sử dụng quy tắc cập nhật đơn giản: $weight = weight - learning_rate * gradient$

Chương trình mẫu:

- **Tập tin:** neural_networks_tutorial.py
- **Nội dung:**

```
# Gọi thư viện
import torch
import torch.nn as nn
import torch.nn.functional as F
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        # 1 input image channel, 6 output channels, 5x5 square convolution
        # kernel
        self.conv1 = nn.Conv2d(1, 6, 5)
        self.conv2 = nn.Conv2d(6, 16, 5)
        # an affine operation: y = Wx + b
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)
    def forward(self, x):
        # Max pooling over a (2, 2) window
        x = F.max_pool2d(F.relu(self.conv1(x)), (2, 2))
        # If the size is a square you can only specify a single number
        x = F.max_pool2d(F.relu(self.conv2(x)), 2)
        x = x.view(-1, self.num_flat_features(x))
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

```

def num_flat_features(self, x):
    size = x.size()[1:] # all dimensions except the batch dimension
    num_features = 1
    for s in size:
        num_features *= s
    return num_features
net = Net()
print(net)

```

Xuất kết quả

```

Net(
  (conv1): Conv2d(1, 6, kernel_size=(5, 5), stride=(1, 1))
  (conv2): Conv2d(6, 16, kernel_size=(5, 5), stride=(1, 1))
  (fc1): Linear(in_features=400, out_features=120, bias=True)
  (fc2): Linear(in_features=120, out_features=84, bias=True)
  (fc3): Linear(in_features=84, out_features=10, bias=True)
)

```

Bạn chỉ cần xác định hàm forward và hàm backward (nơi gradient được tính) được tự động xác định cho bạn sử dụng autograd. Bạn có thể sử dụng bất kỳ hoạt động Tensor nào trong hàm forward.

Các tham số có thể học được của một mô hình được trả về bởi net.parameters()

```

params = list(net.parameters())
print(len(params))
print(params[0].size()) # conv1's .weight

```

Xuất kết quả:

```

10
torch.Size([6, 1, 5, 5])

```

Hãy thử một đầu vào 32x32 ngẫu nhiên: Kích thước đầu vào dự kiến cho mạng này (LeNet) là 32x32. Để sử dụng mạng này trên bộ dữ liệu MNIST, vui lòng thay đổi kích thước hình ảnh từ tập dữ liệu thành 32x32.

```

input = torch.randn(1, 1, 32, 32)
out = net(input)
print(out)

```

Xuất kết quả:

```
tensor([[ 0.1246, -0.0511,  0.0235,  0.1766, -0.0359, -0.0334,  0.1161,
0.0534,
0.0282, -0.0202]], grad_fn=<ThAddmmBackward>)
```

Không có bộ đếm gradient của tất cả các tham số và backprop với các gradient ngẫu nhiên:

```
net.zero_grad()  
out.backward(torch.randn(1, 10))
```

Trước khi tiếp tục, hãy tóm tắt lại tất cả các lớp bạn đã xem cho đến giờ. Tóm tắt:

- + torch.Tensor - Mảng đa chiều với sự hỗ trợ cho các hoạt động của Autograd như backward().
- + nn.Module - Mô-đun mạng nơron. Cách thuận tiện để đóng gói các tham số, với trợ giúp để di chuyển chúng sang GPU, xuất, tải, v.v.
- + nn.Parameter - Một loại Tensor, được tự động đăng ký làm thông số khi được gán làm thuộc tính cho một Module
- + autograd.Function - Thực hiện các định nghĩa về phía trước và phía sau của một hoạt động autograd. Mỗi hoạt động của Tensor tạo ít nhất một Function node, kết nối với các hàm đã tạo Tensor và mã hóa lịch sử của nó.

✚ Hàm Loss

Một hàm loss sẽ lấy cặp đầu vào (đầu ra, đích), và tính giá trị ước lượng khoảng cách đầu ra từ đích.

Có một số chức năng mất khác nhau trong gói package nn. nn.MSELoss tính toán sai số bình phương giữa đầu vào và đích.

Ví dụ :

```
output = net(input)  
target = torch.randn(10) # a dummy target, for example  
target = target.view(1, -1) # make it the same shape as output  
criterion = nn.MSELoss()  
loss = criterion(output, target)  
print(loss)
```

Xuất kết quả :

```
tensor(1.3638, grad_fn=<MseLossBackward>)
```

Bây giờ, nếu bạn theo dõi loss theo hướng lacji hậu, sử dụng .grad_fn , bạn sẽ thấy một biểu đồ tính toán giống như sau:

```
input -> conv2d -> relu -> maxpool2d -> conv2d -> relu -> maxpool2d  
-> view -> linear -> relu -> linear -> relu -> linear  
-> MSELoss  
-> loss
```

Vì vậy, khi gọi loss.backward() , toàn bộ đồ thị được phân biệt và tất cả các Tensors trong biểu đồ có requires_grad=True sẽ có .grad .Tensor tích lũy với gradient.

Để minh họa, chúng ta hãy làm theo một vài bước sau:

```
print(loss.grad_fn) # MSELoss  
print(loss.grad_fn.next_functions[0][0]) # Linear  
print(loss.grad_fn.next_functions[0][0].next_functions[0][0])# ReLU
```

Xuất kết quả:

```
<MseLossBackward object at 0x7f0e86396a90>  
<ThAddmmBackward object at 0x7f0e863967b8>  
<ExpandBackward object at 0x7f0e863967b8>
```

✚ Backdrop

Để truyền lại lỗi, tất cả những gì chúng ta phải làm là để loss.backward() . Bạn cần xóa các gradient hiện có, các gradient khác sẽ được tích lũy thành các gradient hiện có.

Bây giờ chúng ta sẽ gọi loss.backward() và xem xét độ lệch thiên vị của conv1 trước và sau khi lùi.

```
net.zero_grad()    # zeroes the gradient buffers of all parameters  
print('conv1.bias.grad before backward')  
print(net.conv1.bias.grad)  
loss.backward()  
print('conv1.bias.grad after backward')  
print(net.conv1.bias.grad)
```

Xuất kết quả:

```
conv1.bias.grad before backward  
tensor([0., 0., 0., 0., 0., 0.])  
conv1.bias.grad after backward  
tensor([ 0.0181, -0.0048, -0.0229, -0.0138, -0.0088, -0.0107])
```

Cập nhật các Weights

Quy tắc cập nhật đơn giản nhất được sử dụng trong thực tế là Stochastic Gradient Descent (SGD):

$$\text{weight} = \text{weight} - \text{learning_rate} * \text{gradient}$$

Chúng ta có thể thực hiện điều này bằng cách sử dụng mã python đơn giản:

```
learning_rate = 0.01
for f in net.parameters():
    f.data.sub_(f.grad.data * learning_rate)
```

Tuy nhiên, khi bạn sử dụng mạng thần kinh, bạn muốn sử dụng các quy tắc cập nhật khác nhau như SGD, Nesterov-SGD, Adam, RMSProp, v.v. Để kích hoạt, chúng tôi đã xây dựng một gói nhỏ: torch.optim thực hiện tất cả các phương pháp này. Sử dụng nó rất đơn giản:

```
import torch.optim as optim
# create your optimizer
optimizer = optim.SGD(net.parameters(), lr=0.01)
# in your training loop:
optimizer.zero_grad() # không có bộ đệm gradient
output = net(input)
loss = criterion(output, target)
loss.backward()
optimizer.step() # thực hiện cập nhật
```

Lưu ý : Quan sát cách bộ đệm gradient phải được đặt thủ công về 0 bằng cách sử dụng `optimizer.zero_grad()`. Điều này là do gradient được tích lũy như được giải thích trong phần Backprop.

3.5.4. Huấn Luyện Một Phân Loại

➤ Vẽ Dữ Liệu

Thông thường, khi bạn phải xử lý dữ liệu hình ảnh, văn bản, âm thanh hoặc video, bạn có thể sử dụng các gói package python chuẩn để đưa dữ liệu vào một mảng có nhiều mảng. Sau đó, bạn có thể chuyển đổi mảng này thành `torch.*Tensor`.

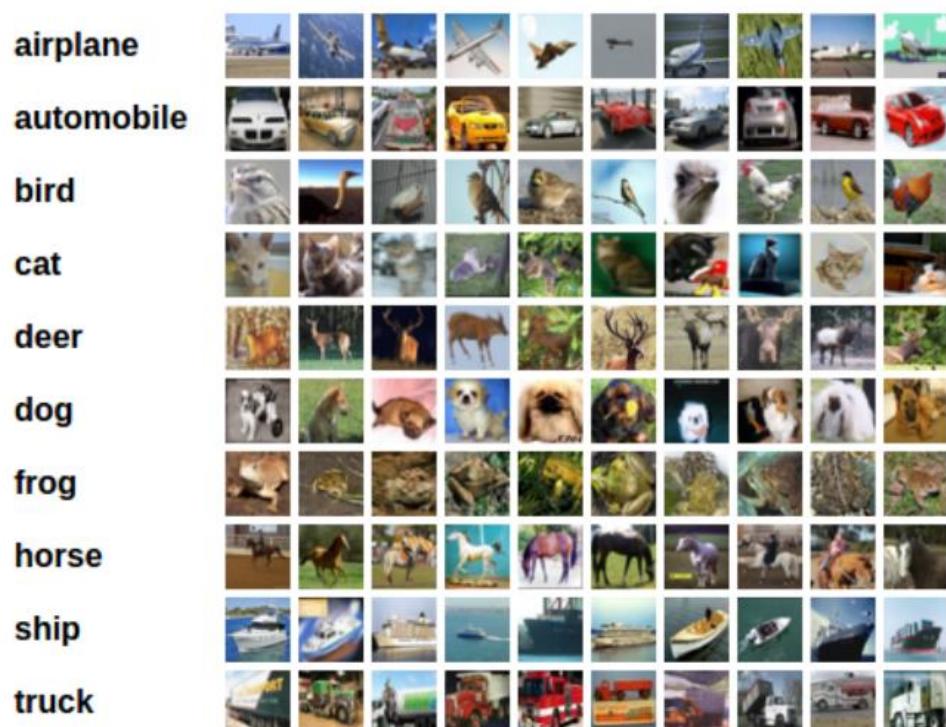
- + Đối với hình ảnh, nên sử dụng các gói như Pillow, OpenCV hữu ích.
- + Đối với âm thanh, nên sử dụng các gói như scipy và librosa.
- + Đối với văn bản, tải Python hoặc Cython gốc, hoặc NLTK và SpaCy là thông dụng.

Cụ thể, chúng tôi đã tạo ra 1 gói với tên torchvision. có bộ tải dữ liệu cho các tập dữ liệu phổ biến như Imagenet, CIFAR10, MNIST, v.v. và máy chuyển dữ liệu cho hình ảnh, viz.,

`torchvision.datasets` and `torch.utils.data.DataLoader`.

Điều này cung cấp một sự thuận tiện rất lớn và tránh viết mã boilerplate.

Đối với hướng dẫn này, chúng tôi sẽ sử dụng tập dữ liệu CIFAR10. Nó có các lớp: ‘airplane’, ‘automobile’, ‘bird’, ‘cat’, ‘deer’, ‘dog’, ‘frog’, ‘horse’, ‘ship’, ‘truck’. Các hình ảnh trong CIFAR-10 có kích thước 3x32x32, tức là hình ảnh màu 3 kênh có kích thước 32x32 pixel.



Hình 3.5. Tập dữ liệu CIFAR10

➤ Huấn luyện phân loại hình ảnh

Ta nên thực hiện các bước sau theo thứ tự sau:

- + Nạp và chuẩn hóa các tập dữ liệu và kiểm tra CIFAR10 bằng cách sử dụng torchvision.
- + Xác định một mạng nơron.
- + Xác định hàm mất.
- + Đào tạo mạng trên dữ liệu đào tạo.
- + Kiểm tra mạng trên dữ liệu thử nghiệm.

➤ *Tải và chuẩn hóa CIFAR10*

Sử dụng torchvision, cực kỳ dễ dàng để tải CIFAR10

```
import torch
import torchvision
import torchvision.transforms as transforms
```

Đầu ra của bộ dữ liệu torchvision là hình ảnh PILImage của phạm vi [0, 1]. Chúng ta biến chúng thành Tensors của phạm vi chuẩn hóa [-1, 1].

```
transform = transforms.Compose(
    [transforms.ToTensor(),
     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])
trainset = torchvision.datasets.CIFAR10(root='./data', train=True,
                                         download=True, transform=transform)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=4,
                                         shuffle=True, num_workers=2)
testset = torchvision.datasets.CIFAR10(root='./data', train=False,
                                         download=True, transform=transform)
testloader = torch.utils.data.DataLoader(testset, batch_size=4,
                                         shuffle=False, num_workers=2)
classes = ('plane', 'car', 'bird', 'cat',
           'deer', 'dog', 'frog', 'horse', 'ship', 'truck')
```

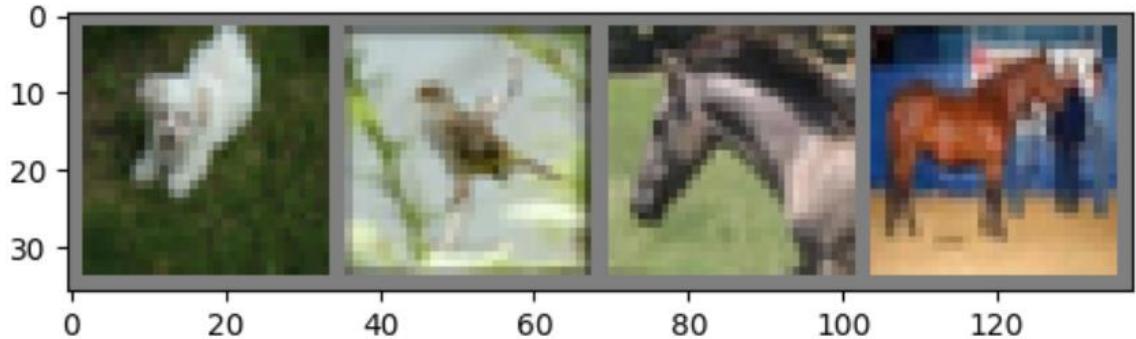
Đầu ra:

```
Downloading https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz to
./data/cifar-10-python.tar.gz
Files already downloaded and verified
```

Chúng ta sẽ hiển thị cho các bạn 1 vài hình ảnh.

```
import matplotlib.pyplot as plt
import numpy as np
# functions to show an image
def imshow(img):
    img = img / 2 + 0.5 # unnormalize
    npimg = img.numpy()
    plt.imshow(np.transpose(npimg, (1, 2, 0)))
# get some random training images
dataiter = iter(trainloader)
images, labels = dataiter.next()
# show images
imshow(torchvision.utils.make_grid(images))
# print labels
```

```
print(''.join('%5s' % classes[labels[j]] for j in range(4)))
```



Đầu ra:

dog bird horse horse

❖ *Xác định một mạng noron*

Sao chép mạng noron từ phần Mạng noron nhân tạo trước và sửa đổi nó để chụp ảnh 3 kênh (thay vì hình ảnh 1 kênh khi nó được xác định).

```
import torch.nn as nn
import torch.nn.functional as F
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)
    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 16 * 5 * 5)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
```

```
x = self.fc3(x)
return x
net = Net()
```

❖ Xác định hàm Loss và trình tối ưu hóa

```
import torch.optim as optim
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)
```

❖ Huấn luyện mạng

Đây là khi mọi thứ bắt đầu trở nên thú vị. Chúng ta chỉ cần lặp qua trình lặp dữ liệu của chúng ta và nạp các đầu vào vào mạng và tối ưu hóa.

```
for epoch in range(2): # loop over the dataset multiple times
    running_loss = 0.0
    for i, data in enumerate(trainloader, 0):
        # get the inputs
        inputs, labels = data
        # zero the parameter gradients
        optimizer.zero_grad()
        # forward + backward + optimize
        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        # print statistics
        running_loss += loss.item()
        if i % 2000 == 1999: # print every 2000 mini-batches
            print('[%d, %5d] loss: %.3f' %
                  (epoch + 1, i + 1, running_loss / 2000))
            running_loss = 0.0
    print('Finished Training')
```

Đầu ra:

```
[1, 2000] loss: 2.236
[1, 4000] loss: 1.880
[1, 6000] loss: 1.676
[1, 8000] loss: 1.586
[1, 10000] loss: 1.515
[1, 12000] loss: 1.464
[2, 2000] loss: 1.410
```

```
[2, 4000] loss: 1.360  
[2, 6000] loss: 1.360  
[2, 8000] loss: 1.325  
[2, 10000] loss: 1.312  
[2, 12000] loss: 1.302  
Finished Training
```

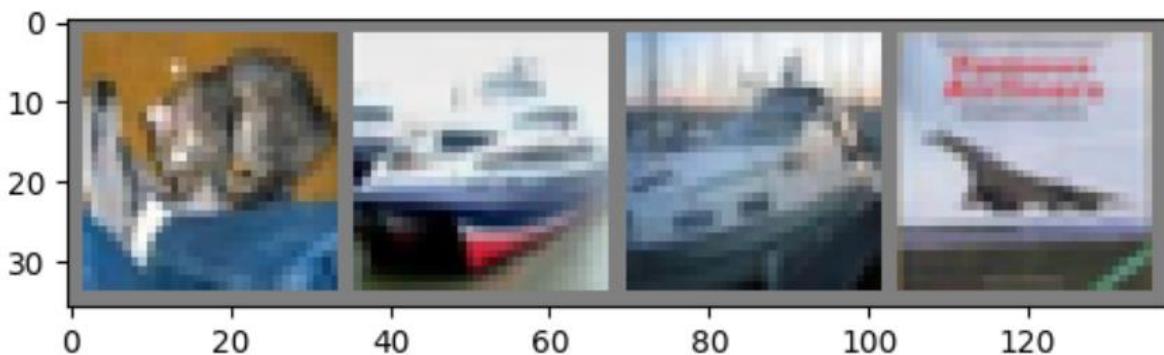
➤ Kiểm tra mạng

Chúng tôi đã đào tạo mạng cho 2 lần vượt qua tập dữ liệu đào tạo. Nhưng chúng ta cần kiểm tra xem mạng có học được gì không.

Chúng tôi sẽ kiểm tra điều này bằng cách dự đoán nhãn lớp mà mạng nơ-ron xuất ra và kiểm tra nó dựa trên sự thật. Nếu dự đoán là chính xác, chúng tôi thêm mẫu vào danh sách các dự đoán chính xác.

Bước đầu tiên. Hãy để chúng tôi hiển thị một hình ảnh từ bộ thử nghiệm để làm quen.

```
dataiter = iter(testloader)  
images, labels = dataiter.next()  
# print images  
imshow(torchvision.utils.make_grid(images))  
print('GroundTruth: ', ' '.join('%5s' % classes[labels[j]] for j in range(4)))
```



Đầu ra:

GroundTruth: cat ship ship plane

Bây giờ chúng ta hãy xem mạng nơron nghĩ những ví dụ trên là:

```
outputs = net(images)
```

Các đầu ra là năng lượng cho 10 lớp. Nâng cao năng lượng cho một lớp, mạng càng nghĩ rằng hình ảnh là của một lớp cụ thể. Vì vậy, hãy lấy chỉ số năng lượng cao nhất:

```

_, predicted = torch.max(outputs, 1)
print('Predicted: ', ''.join('%5s' % classes[predicted[j]]
                             for j in range(4)))

```

Đầu ra:

```
Predicted: cat car car plane
```

Chúng ta hãy xem cách mạng hoạt động trên toàn bộ tập dữ liệu.

```

correct = 0
total = 0
with torch.no_grad():
    for data in testloader:
        images, labels = data
        outputs = net(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()
print('Accuracy of the network on the 10000 test images: %d %%' % (
    100 * correct / total))

```

Đầu ra:

```
Accuracy of the network on the 10000 test images: 54 %
```

Điều đó có vẻ theo hướng tốt hơn, đó là độ chính xác 10% (ngẫu nhiên chọn một lớp trong số 10 lớp). Có vẻ như mạng đã học được điều gì đó.

Các lớp hoạt động tốt và các lớp không hoạt động tốt:

```

class_correct = [0. for i in range(10)]
class_total = [0. for i in range(10)]
with torch.no_grad():
    for data in testloader:
        images, labels = data
        outputs = net(images)
        _, predicted = torch.max(outputs, 1)
        c = (predicted == labels).squeeze()
        for i in range(4):
            label = labels[i]
            class_correct[label] += c[i].item()
            class_total[label] += 1
for i in range(10):
    print('Accuracy of %5s : %2d %%' % (

```

```
classes[i], 100 * class_correct[i] / class_total[i]))
```

Đầu ra:

```
Accuracy of plane : 61 %
Accuracy of car : 85 %
Accuracy of bird : 46 %
Accuracy of cat : 23 %
Accuracy of deer : 40 %
Accuracy of dog : 36 %
Accuracy of frog : 80 %
Accuracy of horse : 59 %
Accuracy of ship : 65 %
Accuracy of truck : 46 %
```

➤ *Huấn luyện Trên GPU*

Cũng giống như cách bạn chuyển Tensor sang GPU, bạn chuyển mạng neural lên GPU.

Trước tiên, hãy xác định thiết bị của chúng tôi là thiết bị cuda hiển thị đầu tiên nếu chúng tôi có CUDA:

```
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
# Assume that we are on a CUDA machine, then this should print a
# CUDA device:
print(device)
```

Đầu ra:

```
cuda:0
```

Phần còn lại của phần này giả định rằng thiết bị là một thiết bị CUDA.

Sau đó, các phương thức này sẽ đi qua tất cả các mô-đun và chuyển đổi các tham số và bộ đệm của chúng thành các dải CUDA:

```
net.to(device)
```

Hãy nhớ rằng bạn sẽ phải gửi các đầu vào và mục tiêu ở mọi bước tới GPU:

```
inputs, labels = inputs.to(device), labels.to(device)
```

CHƯƠNG 4. XÂY DỰNG ỨNG DỤNG

4.1. Bài toán xây dựng bộ phân lớp hình ảnh

Các phương pháp tiếp cận hiện nay để nhận dạng đối tượng sử dụng thiết yếu các phương pháp học máy. Để cải thiện hiệu suất của chúng, chúng tôi có thể thu thập các bộ dữ liệu lớn hơn, tìm hiểu các mô hình mạnh hơn và sử dụng tốt hơn kỹ thuật ngăn ngừa quá mức. Cho đến gần đây, bộ dữ liệu của hình ảnh được dán nhãn là tương đối nhỏ - theo thứ tự hàng chục ngàn hình ảnh (ví dụ: NORB [16], Caltech-101/256 [8, 9] và CIFAR-10/100 [12]). Các tác vụ nhận dạng đơn giản có thể được giải quyết khá tốt với các bộ dữ liệu có kích thước này, đặc biệt là nếu chúng được tăng cường với các biến đổi bảo quản nhãn. Ví dụ, hiện tại tỷ lệ lỗi trên tác vụ nhận dạng chữ số MNIST (<0,3%) tiếp cận hiệu suất của con người. Nhưng các đối tượng trong các thiết lập thực tế thể hiện sự thay đổi đáng kể, vì vậy để học cách nhận ra chúng, đó là cần thiết để sử dụng bộ đào tạo lớn hơn nhiều.

Và thực sự, những thiếu sót của bộ dữ liệu hình ảnh nhỏ đã được công nhận rộng rãi (ví dụ, Pinto và cộng sự), nhưng chỉ gần đây mới có thể thu thập được bộ dữ liệu được dán nhãn với hàng triệu hình ảnh. Các bộ dữ liệu mới lớn hơn bao gồm LabelMe , trong đó bao gồm hàng trăm ngàn hình ảnh được phân đoạn đầy đủ và ImageNet, bao gồm hơn 15 triệu hình ảnh có độ phân giải cao được dán nhãn trong hơn 22.000 danh mục. Để tìm hiểu về hàng ngàn đối tượng từ hàng triệu hình ảnh, chúng ta cần một mô hình với lượng học tập lớn sức chứa. Tuy nhiên, sự phức tạp to lớn của nhiệm vụ nhận dạng đối tượng có nghĩa là vấn đề này không thể được chỉ định ngay cả bởi một tập dữ liệu lớn như Image Net, vì vậy mô hình của chúng tôi cũng nên có nhiều về kiến thức trước để bù đắp cho tất cả dữ liệu chúng tôi không có.

Mạng lưới thần kinh chuyển đổi (DCNNs) tạo thành một loại mô hình như vậy [16, 11, 13, 18, 15, 22, 26]. Năng lực của họ có thể được kiểm soát bằng cách thay đổi độ sâu và chiều rộng của chúng, và chúng cũng đưa ra các giả định mạnh mẽ và chủ yếu là chính xác về bản chất của hình ảnh (cụ thể là sự ổn định của thống kê và địa phương của các phụ thuộc pixel). Do đó, so với các mạng nơ ron phản hồi tiêu chuẩn với các lớp có kích thước tương tự nhau, CNN có ít kết nối và tham số hơn và do đó chúng dễ đào tạo hơn, trong khi tốt nhất về mặt lý thuyết hiệu suất có thể chỉ kém hơn một chút.

Mặc dù chất lượng hấp dẫn của DCNN và mặc dù hiệu quả tương đối của kiến trúc địa phương của họ, chúng vẫn còn quá đắt để áp dụng ở quy mô lớn cho hình ảnh có độ phân giải cao. May mắn thay, GPU hiện tại, được kết hợp với triển khai tích hợp 2D được tối ưu hóa cao, rất mạnh mẽ đủ để tạo điều kiện cho việc đào tạo các DCNN lớn thú vị và các bộ dữ liệu gần đây như ImageNet chứa đủ các ví dụ được dán nhãn để huấn luyện các mô hình như vậy mà không bị quá tải nghiêm trọng.

4.2. Giới thiệu tập dữ liệu CIFAR10 và trình bày các bước đã thực hiện để lấy tập dữ liệu từ tập CIFAR10

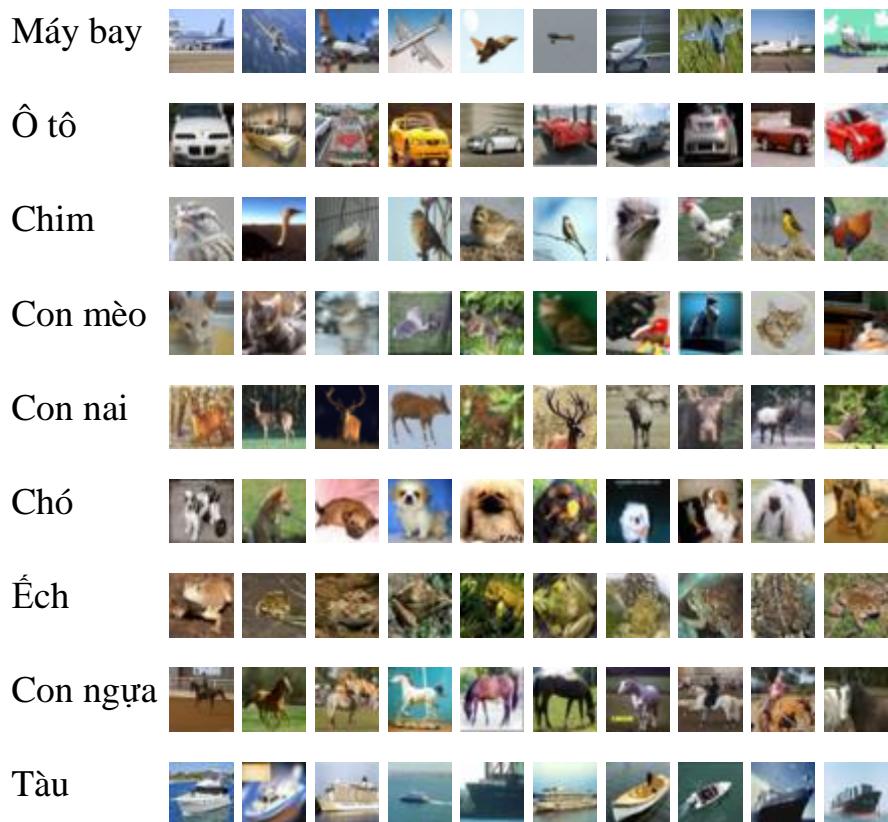
4.2.1. Giới thiệu tập dữ liệu CIFAR10

Tập dữ liệu CIFAR10 (CIFAR-10 dataset) là một tập dữ liệu có gán nhãn được thu thập bởi Alex Krizhevsky, Vinod Nair và Geoffrey Hinton. CIFAR10 là tập con của tập dữ liệu “80 million tiny images”.

Bộ dữ liệu CIFAR-10 bao gồm 60000 hình ảnh màu 32x32 trong 10 lớp, với 6000 hình ảnh mỗi lớp. Có 50000 hình ảnh đào tạo và 10000 hình ảnh thử nghiệm.

Bộ dữ liệu được chia thành năm đợt đào tạo và một đợt thử nghiệm, mỗi đợt có 10000 hình ảnh. Lô thử nghiệm chứa chính xác 1000 hình ảnh được chọn ngẫu nhiên từ mỗi lớp. Các lô đào tạo chứa các hình ảnh còn lại theo thứ tự ngẫu nhiên, nhưng một số lô đào tạo có thể chứa nhiều hình ảnh từ một lớp hơn một lớp khác. Giữa chúng, các đợt đào tạo chứa chính xác 5000 hình ảnh từ mỗi lớp.

Đây là một bộ cơ sở dữ liệu tương đối khó vì ảnh nhỏ và object trong cùng một class cũng biến đổi rất nhiều về màu sắc, hình dáng, kích thước. Thuật toán tốt nhất hiện nay cho bài toán này đã đạt được độ chính xác trên 90%, sử dụng một Convolutional Neural Network nhiều lớp kết hợp với Softmax regression ở layer cuối cùng.



Xe tải



Hình 4.1: Tập dữ liệu CIFAR10

Tập dữ liệu (frog, horse, ship, truck) được lấy hoàn toàn từ tập dữ liệu CIFAR10. Do đó:

- + Gồm 4 lớp, mỗi lớp có 6000 ảnh
- + Tập train: 20000 ảnh, chia đều cho 4 lớp (mỗi lớp 5000 ảnh).
- + Tập test: 4000 ảnh, chia đều cho 4 lớp (mỗi lớp 1000 ảnh).

4.2.2. Các bước đã thực hiện để lấy tập dữ liệu từ tập CIFAR10

+ **B1:** Chuyển dữ liệu thô CIFAR10 (được download và chạy với tập dữ liệu CIFAR10). Chạy chương trình chuyển dữ liệu thô CIFAR10 tại <https://github.com/knjcode/cifar2png>

+ **B2:** Sau khi chạy chương trình, dữ liệu CIFAR10 sẽ được chuyển thành các tập tin hình ảnh, và được lưu trong thư mục, ứng với tập train, tập test và theo từng lớp.

+ **B3:** Chọn 4 lớp dữ liệu cần dùng cho ứng dụng của nhóm, có thể tổ chức lại nếu cần.

+ **B4:** Sử dụng `torchvision.datasets.ImageFolder(...)` của thư viện PyTorch tạo tập dữ liệu (`train`, `test`) từ dữ liệu đã chọn

```
train_data = torchvision.datasets.ImageFolder (root='./data/train',
                                              transform=transform)
train_data_loader = torch.utils.data.DataLoader(train_data,
                                                batch_size=size_of_batch, shuffle=True,num_workers=0)
test_data = torchvision.datasets.ImageFolder(root='./data/test',
                                              transform=transform)
test_data_loader = torch.utils.data.DataLoader(test_data,
                                               batch_size=size_of_batch, shuffle=True,num_workers=0)
classes =("frog", "horse", "ship", "truck")
```

4.3. Phương pháp lựa chọn của đề tài để xây dựng bộ phân lớp hình ảnh (Mạng nơ-ron tích chập – DCNN (Deep Convolution Neural Network))

Với sự phát triển phồn chúng mạnh mẽ cho phép tính toán song song hàng tỉ phép tính, tạo tiền đề cho Mạng nơ-ron tích chập trở nên phổ biến và đóng vai trò quan trọng trong sự phát triển của trí tuệ nhân tạo nói chung và xử lý ảnh nói riêng. Một trong các ứng dụng quan trọng của mạng nơ-ron tích chập đó là cho phép các máy tính có khả năng “nhìn” và “phân tích”, nói 1 cách dễ hiểu, Convnets được sử dụng để nhận dạng hình ảnh bằng cách đưa

nó qua nhiều layer với một bộ lọc tích chập để sau cùng có được một điểm số nhận dạng đối tượng. DCNN được lấy cảm hứng từ vỏ não thị giác.

Mỗi khi chúng ta nhìn thấy một cái gì đó, một loạt các lớp tế bào thần kinh được kích hoạt, và mỗi lớp thần kinh sẽ phát hiện một tập hợp các đặc trưng như đường thẳng, cạnh, màu sắc, v.v.v của đối tượng. lớp thần kinh càng cao sẽ phát hiện các đặc trưng phức tạp hơn để nhận ra những gì chúng ta đã thấy.

ConvNet có 02 phần chính: Lớp trích lọc đặc trưng của ảnh (Conv, Relu và Pool) và Lớp phân loại (FC và softmax).

Đầu vào (dữ liệu training): Input đầu vào là một bức ảnh được biểu diễn bởi ma trận pixel với kích thước: [w x h x d]

- W: chiều rộng
- H: chiều cao
- D: Là độ sâu, hay dễ hiểu là số lớp màu của ảnh. Ví dụ ảnh RBG sẽ là 3 lớp ảnh Đỏ, Xanh Dương, Xanh.

- *Conv Layer:*

Mục tiêu của các lớp tích chập là trích chọn các đặc trưng của ảnh đầu vào. Ảnh đầu vào được cho qua một bộ lọc chạy dọc bức ảnh. Bộ lọc có kích thước là (3x3 hoặc 5x5) và áp dụng phép tích vô hướng để tính toán, cho ra một giá trị duy nhất. Đầu ra của phép tích chập là một tập các giá trị ảnh được gọi là mạng đặc trưng (features map).

Thực chất, ở các layer đầu tiên, phép tích chập đơn giản là phép tìm biên ảnh, nếu các bạn có kiến thức cơ bản về xử lý ảnh. Còn nếu không thì bạn chỉ cần hiểu sau khi cho qua bộ lọc nó sẽ làm hiện lên các đặc trưng của đối tượng trong ảnh như đường vẽ xung quanh đối tượng, các góc cạnh, v.v., và các layer tiếp theo sẽ lại trích xuất tiếp các đặc trưng của đặc trưng của các đối tượng đó, việc có nhiều layer như vậy cho phép chúng ta chia nhỏ đặc trưng của ảnh tới mức nhỏ nhất có thể. Vì thế mới gọi là mạng đặc trưng.

- *ReLU Layer:*

ReLU layer áp dụng các kích hoạt (activation function) $\max(0, x)$ lên đầu ra của Conv Layer, có tác dụng đưa các giá trị âm về thành 0. Layer này không thay đổi kích thước của ảnh và không có thêm bất kì tham số nào.

Mục đích của lớp ReLU là đưa ảnh một mức ngưỡng, ở đây là 0. Để loại bỏ các giá trị âm không cần thiết mà có thể sẽ ảnh hưởng cho việc tính toán ở các layer sau đó.

- *Pool Layer:*

Pool Layer thực hiện chức năng làm giảm chiều không gian của đầu và giảm độ phức tạp tính toán của model ngoài ra Pool Layer còn giúp kiểm soát hiện tượng overfitting. Thông thường, Pool layer có nhiều hình thức khác nhau phù hợp cho nhiều bài toán, tuy nhiên Max Pooling là được sử dụng nhiều vào phổ biến hơn cả với ý tưởng cũng rất sát với thực tế con người đó là: Giữ lại chi tiết quan trọng hay hiểu ở trong bài toán này chính giữ lại pixel có giá trị lớn nhất.

Thông thường max pooling có kích thước là 2 và stride=2. Nếu lấy giá trị quá lớn, thay vì giảm tính toán nó lại làm phá vỡ cấu trúc ảnh và mất mát thông tin nghiêm trọng. Vì vậy mà một số chuyên gia không thích sử dụng layer này mà thay vào đó sử dụng thêm các lớp Conv Layer và tăng số stride lên mỗi lần.

- *Fully_Connected Layer (FC):*

Tên tiếng việt là Mạng liên kết đầy đủ. Tại lớp mạng này, mỗi một nơ-ron của layer này sẽ liên kết tới mọi nơ-ron của lớp khác. Để đưa ảnh từ các layer trước vào mạng này, buộc phải dàn phẳng bức ảnh ra thành 1 vector thay vì là mảng nhiều chiều như trước. Tại layer cuối cùng sẽ sử dụng 1 hàm kinh điển trong học máy mà bất kì ai cũng từng sử dụng đó là softmax để phân loại đối tượng dựa vào vector đặc trưng đã được tính toán của các lớp trước đó.

4.4. Một số lớp cơ bản của 1 DCNN được sử dụng trong ứng dụng (Deep Convolution Neural Network)

4.4.1. Lớp tích chập:

```
class torch.nn.Conv2d(in_channels, out_channels, kernel_size,
stride=1, padding=0, dilation=1, groups=1, bias=True)
```

Áp dụng tích chập 2 chiều trên một tín hiệu đầu vào bao gồm nhiều mặt phẳng đầu vào.

Trong trường hợp đơn giản nhất, giá trị đầu ra của lớp có kích thước đầu vào (N, C_{in}, H, W) và đầu ra ($N, C_{out}, H_{out}, W_{out}$) có thể được mô tả cụ thể như sau:

$$out(N_i, C_{out_j}) = bias(C_{out_j}) + \sum_{k=0}^{C_{in}-1} weight(C_{out_j}, k) * input(N_i, k)$$

Trong đó $*$ là toán tử tương quan chéo 2 chiều có giá trị, N là kích thước batch, C biểu thị số lượng kênh, H là chiều cao của các mặt phẳng đầu vào tính bằng pixel và W là chiều rộng tính bằng pixel.

- **stride:** kiểm soát bước nhảy cho sự tương quan chéo, là 1 số đơn hoặc 1 cặp số.

- **padding**: kiểm soát số lượng zero-paddings ẩn trên cả hai mặt với số điểm **padding** cho mỗi kích thước.
- **dilation**: kiểm soát khoảng cách giữa các điểm kernel
- **group**: kiểm soát các kết nối giữa đầu vào và đầu ra. **in_channels** và **out_channels** cả hai đều phải được chia ra bởi **group**. Ví dụ:
 - Khi group = 1, tất cả các đầu vào được kết hợp với tất cả các đầu ra.
 - Khi group = 2, quá trình hoạt động trở nên tương đương với việc có hai lớp đối diện cạnh nhau, mỗi lớp nhìn thấy một nửa kênh đầu vào và tạo ra một nửa kênh đầu ra và cả hai sau đó được ghép nối lại với nhau.
 - Khi groups= **in_channels** mỗi kênh đầu vào được kết hợp với bộ lọc riêng của nó (với size $\left\lfloor \frac{\text{out_channels}}{\text{in_channels}} \right\rfloor$).

Các thông số **kernel_size**, **stride**, **padding**, **dilation** có thể là:

- 1 số đơn int – sử dụng trong trường hợp có cùng một giá trị được sử dụng cho chiều cao và chiều rộng.
- 1 cặp của hai số int - trong trường hợp này, int đầu tiên được sử dụng cho chiều cao và int thứ hai cho chiều rộng

*Chú ý:

Tùy thuộc vào kích thước của kernel, một số cột (cuối cùng) của đầu vào có thể bị mất, bởi vì nó là một sự tương quan chéo hợp lệ, và không phải là một tương quan chéo hoàn toàn. Điều đó dẫn đến người dùng có thể thêm phần đệm (padding) thích hợp.

*Chú ý:

Cấu hình khi $\text{groups} == \text{in_channels}$ và $\text{out_channels} == K * \text{in_channels}$ trong đó K là một số nguyên dương và theo văn chương thì được gọi là tích chập theo chiều sâu

Nói cách khác, đối với một đầu vào có kích thước ($N, C_{in}, H_{in}, W_{in}$), nếu bạn muốn tích chập theo chiều sâu với một số K theo chiều sâu, thì bạn sử dụng các đối số hàm tạo.

($\text{in_channels}=C_{in}$, $\text{out_channels}=C_{in}*K, \dots$, $\text{groups}=C_{in}$)

Các thông số:

- **in_channels** (int) - Số kênh trong ảnh đầu vào
- **out_channels** (int) - Số kênh được tạo ra bởi tích chập
- **kernel_size** (int hoặc tuple) - Kích thước của kernel tích chập
- **stride** (int hoặc tuple, tùy chọn) - bước nhảy của tích chập. Mặc định: 1

- **padding** (int hoặc tuple, tùy chọn) - Zero-padding được thêm vào cả hai bên của đầu vào. Mặc định: 0
- **dilation** (int hoặc tuple, tùy chọn) - Khoảng cách giữa các phần tử kernel. Mặc định: 1
- **groups** (int, tùy chọn) - Số lượng kết nối bị chặn từ kênh đầu vào đến kênh đầu ra. Mặc định: 1
- **bias** (bool, tùy chọn) - Nếu True, thêm một bias có thể học được vào đầu ra. Mặc định: True

Công thức:

- Đầu vào: $(N, C_{in}, H_{in}, W_{in})$
- Đầu ra: $(N, C_{out}, H_{out}, W_{out})$ trong đó:

$$H_{out} = \left\lfloor \frac{H_{in} + 2 \times \text{padding}[0] \times \text{dilation}[0] \times (\text{kernel_size}[0] - 1) - 1}{\text{stride}[0]} + 1 \right\rfloor$$

$$W_{out} = \left\lfloor \frac{W_{in} + 2 \times \text{padding}[1] \times \text{dilation}[1] \times (\text{kernel_size}[1] - 1) - 1}{\text{stride}[1]} + 1 \right\rfloor$$

Biết:

- weight (Tensor) - trọng số có thể học được của mô-đun hình dạng (out_channels, in_channels, kernel_size [0], kernel_size [1])
- bias (Tensor) - bias có thể học được của mô-đun hình dạng (out_channels)

4.4.2. Lớp pooling:

```
class torch.nn.MaxPool2d(kernel_size, stride=None, padding=0,
dilation=1, return_indices=False, ceil_mode=False)
```

Áp dụng pooling tối đa 2 chiều trên tín hiệu đầu vào bao gồm nhiều mặt phẳng đầu vào.

Trong trường hợp đơn giản nhất, giá trị đầu ra của lớp có kích thước đầu vào (N, C, H, W) , đầu ra (N, C, H_{out}, W_{out}) và **kernel_size** (kH, kW) có thể được mô tả cụ thể như sau:

$$\text{out}(N_i, C_i, h, w) = \max_{m=0, \dots, kH-1} \max_{n=0, \dots, kW-1} \text{input}(N_i, C_i, \text{stride}[0] * h + m, \text{stride}[1] * w + n)$$

Nếu **padding** khác 0, thì đầu vào được chèn 0 hoàn toàn ở cả hai bên cho số điểm **padding**. **dilation** kiểm soát khoảng cách giữa các điểm kernel.

Các tham số **kernel_size**, **stride**, **padding**, **dilation** có thể là:

- 1 số đơn int – sử dụng trong trường hợp có cùng một giá trị được sử dụng cho chiều cao và chiều rộng.

- 1 cặp của hai số int - trong trường hợp này, int đầu tiên được sử dụng cho chiều cao và int thứ hai cho chiều rộng

Các thông số:

- **kernel_size** - kích thước của cửa sổ để có thể lấy tối đa
- **stride** – bước nhảy của cửa sổ. Giá trị mặc định là kernel_size
- **padding** - chèn 0 ẩn để được thêm vào cả hai bên
- **dilation** - một tham số kiểm soát bước nhảy của các phần tử trong cửa sổ
- **return_indices** - nếu True, sẽ trả về các chỉ mục tối đa cùng với các kết quả đầu ra. Hữu ích khi Unpooling sau
- **ceil_mode** - khi True, sẽ sử dụng *ceil* thay thế cho *floor* để tính toán hình dạng đầu ra

Công thức:

- Đầu vào: (N, C, H_{in}, W_{in})
- Đầu ra: (N, C, H_{out}, W_{out}) trong đó:

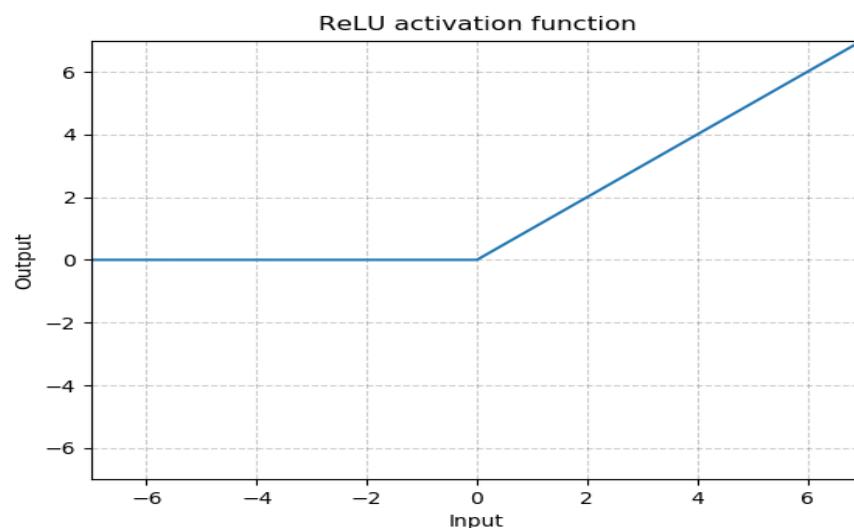
$$H_{out} = \left\lfloor \frac{H_{in} + 2 \times \text{padding}[0] \times \text{dilation}[0] \times (\text{kernel_size}[0] - 1) - 1}{\text{stride}[0]} + 1 \right\rfloor$$

$$W_{out} = \left\lfloor \frac{W_{in} + 2 \times \text{padding}[1] \times \text{dilation}[1] \times (\text{kernel_size}[1] - 1) - 1}{\text{stride}[1]} + 1 \right\rfloor$$

4.4.3. Hàm truyền (activation function):

torch.nn.functional.relu(input, inplace=False) → Tensor

Áp dụng hàm ReLU ($x = \max(0, x)$)



Hình 4.2 Hàm truyền

Tham số: **inplace** - có thể tùy chọn thực hiện thao tác tại chỗ. Mặc định: False

Công thức:

- Đầu vào: ($N, *$), trong đó * có nghĩa là bất kỳ con số nào.
- Đầu ra: ($N, *$), trong đó * giống với đầu vào.

4.4.4. Lớp dropout:

```
class torch.nn.Dropout2d(p=0.5, inplace=False)
```

Tự động 0 toàn bộ các kênh của tensor đầu vào. Các kênh thành 0 được ngẫu nhiên trên mỗi lần gọi chuyển tiếp.

Thông thường đầu vào đến từ mô-đun **nn.Conv2d**.

Như được mô tả trong bài báo **Efficient Object Localization Using Convolutional Networks**, nếu các điểm ảnh lân cận trong bản đồ tính năng có tương quan chặt chẽ (như là trường hợp bình thường trong các lớp tích chập sớm) sau đó i.i.d. dropout sẽ không thường xuyên có sự kích hoạt và nếu không sẽ chỉ dẫn đến kết quả là giảm tỷ lệ học hiệu quả.

Trong trường hợp này, **nn.Dropout2d ()** sẽ giúp thúc đẩy tính độc lập giữa các bản đồ tính năng và nên được sử dụng thay thế.

Các thông số:

- **p** (float, optional) - xác suất của một phần tử được đổi thành 0
- **inplace** (bool, tùy chọn) - Nếu được đặt thành True, sẽ thực hiện thao tác in-place

Công thức:

- Đầu vào: (N, C, Hi, W)
- Đầu ra: (N, C, H, W)

4.4.5. Lớp liên kết đầy đủ (fully connected layer):

```
class torch.nn.Linear(in_features, out_features, bias=True)
```

Áp dụng phép chuyển đổi tuyến tính cho dữ liệu đến: $y = xA^T + b$

Các thông số:

- **in_features** - kích thước của mỗi mẫu đầu vào
- **out_features** - kích thước của mỗi mẫu đầu ra
- **bias** - Nếu được đặt thành False, lớp sẽ học bias thêm vào. Mặc định: True

Công thức:

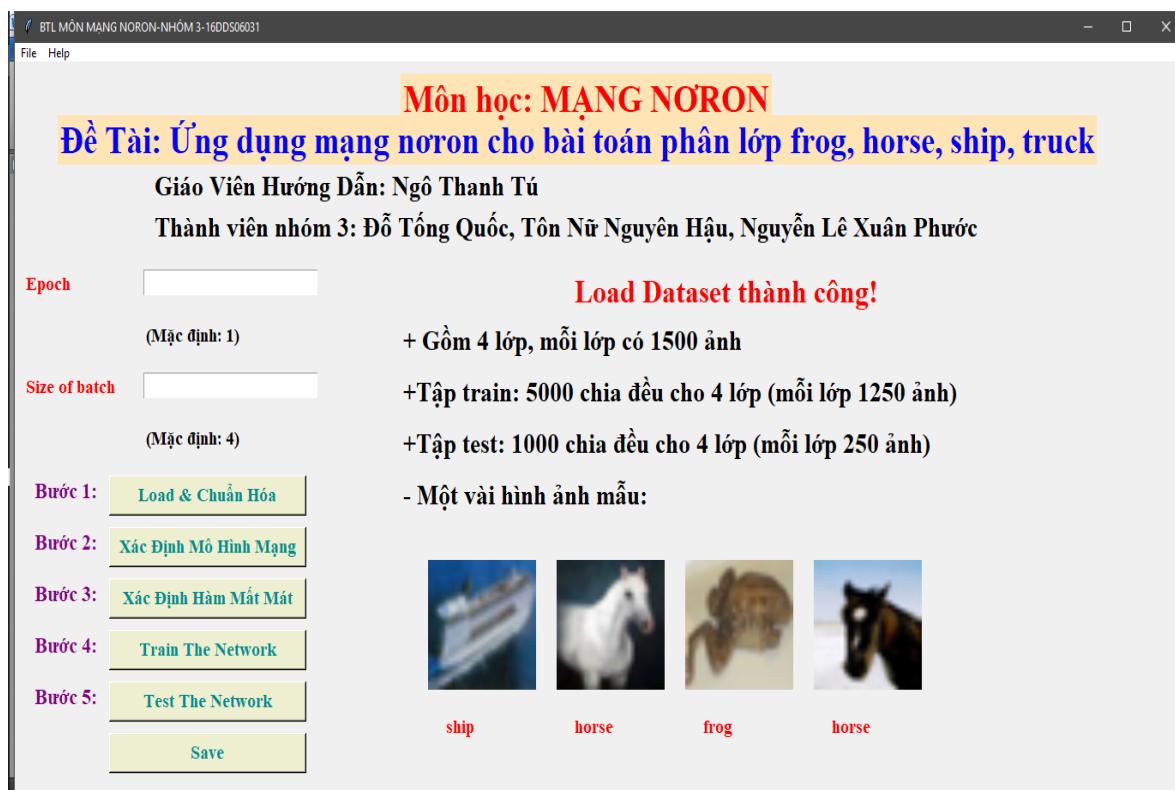
- Đầu vào: ($N, *, in_features$), trong đó * có nghĩa là bất kỳ nào
- Đầu ra: ($N, *, out_features$), trong đó tất cả các kích thước trừ kích thước cuối cùng có cùng dạng với đầu vào.

4.4. Giao diện, các chức năng của ứng dụng xây dựng bộ phân lớp hình ảnh bằng mạng DCNN



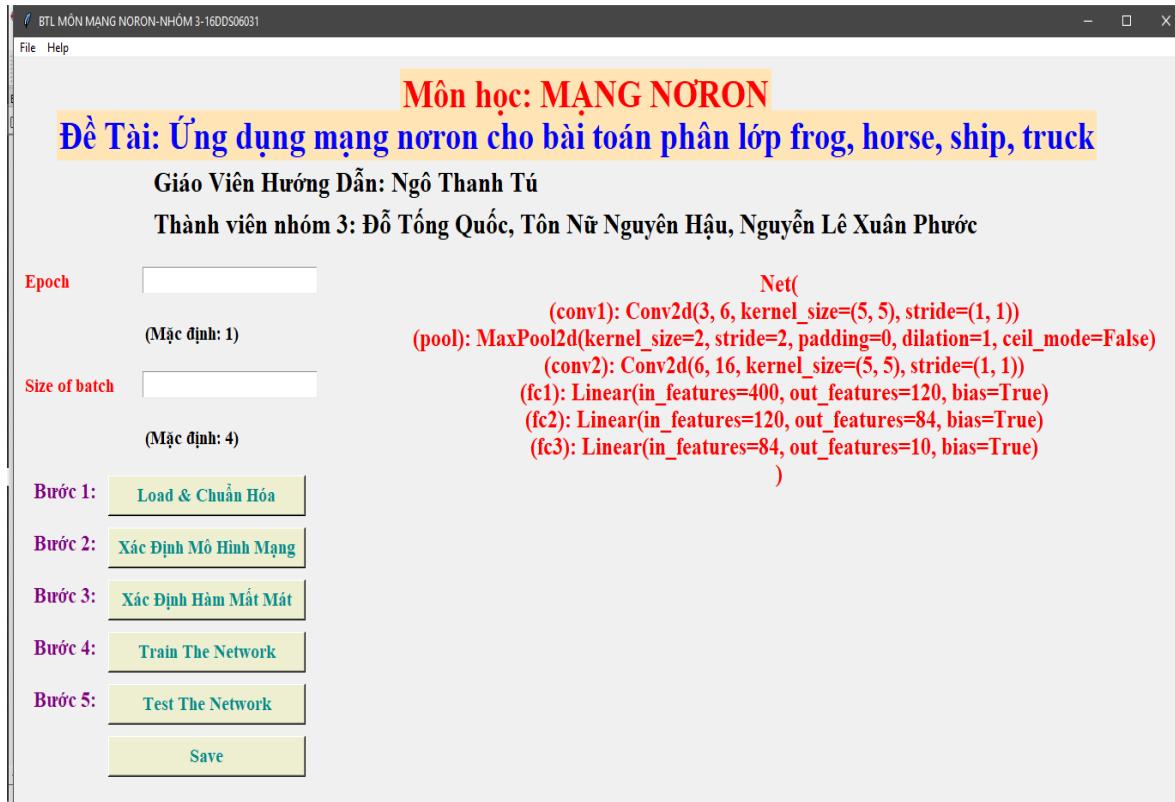
Hình 4.3: Giao diện chính chương trình

Bước 1: Load và chuẩn hóa các tập dữ liệu huấn luyện/ kiểm tra



Hình 4.4: Load và chuẩn hóa các tập dữ liệu huấn luyện/ kiểm tra

Bước 2: Xác định mô hình mạng noron



Hình 4.5: Mô hình mạng noron

Bước 3: Xác định hàm mất mát



Hình 4.6 Hàm mất mát và tối ưu hóa

Bước 4: Huấn luyện mạng trên tập dữ liệu huấn luyện.



Hình 4.7: Kết quả Huấn luyện mạng trên tập dữ liệu huấn luyện

Bước 5: Kiểm tra mạng trên tập dữ liệu kiểm tra.



Hình 4.8: Kết quả Kiểm tra mạng trên tập dữ liệu kiểm tra

4.5. Hướng dẫn sử dụng

Bước 1: Click vào nút Load & Chuẩn Hóa để load Dataset huấn luyện, sau khi load thành công Dataset màn hình sẽ hiển thị: Số lớp trong Dataset, tên của các lớp, số mẫu trong mỗi tập và hiển thị hình ảnh minh họa cho mỗi tập.

Bước 2 & 3: Sau khi load Dataset thành công ở bước 1, click vào nút "Xác Định Mô Hình Mạng" để màn hình sẽ hiển thị mô hình mạng nơron tiếp tục click vào nút "Xác Định Hàm Mất Mát", màn hình sẽ hiển thị hàm mất mát và tối ưu

Bước 4: Tại bước này, bạn có thể thay đổi một số thông số huấn luyện theo ý muốn. Sau khi kiểm tra click vào nút Train the Network, nhập tên để lưu lại mô hình mạng sau khi huấn luyện, sau đó quá trình huấn luyện sẽ bắt đầu. Khi quá trình huấn luyện hoàn tất sẽ có thông báo đường dẫn chứa file lưu lại

Bước 5: Click vào nút Test the Network và chọn file mô hình mạng đã được lưu trước đó, sau khi test hoàn tất ứng dụng sẽ hiển thị kết quả kiểm tra

KẾT LUẬN

1. *Những đóng góp của bài tập lớn:*

Qua nghiên cứu và thực nghiệm, bài tập lớn đã đạt được những kết quả chính như sau:

- Nghiên cứu tổng quan về mạng nơron, deep learning và một số ứng dụng của deep learning.
- Nghiên cứu chi tiết mô hình học sâu tiêu biểu là DCNN.
- Ứng dụng các kỹ thuật đã tìm hiểu để giải quyết bài toán phân lớp hình ảnh thông qua mô hình huấn luyện DCNN.
- Đã tiến hành thiết kế ứng dụng để huấn luyện và kiểm tra trên tập dữ liệu.

2. *Hướng phát triển:*

Việc cải tiến mô hình DCNN và ứng dụng mô hình vào thực tế vẫn đang được các nhà nghiên cứu quan tâm và xây dựng. Bài tập lớn này tuy đạt được một số kết quả nếu trên nhưng vẫn còn nhiều hạn chế do điều kiện về mặt thời gian và phạm vi nghiên cứu của đề tài. Vì vậy, hướng nghiên cứu tiếp theo của ứng dụng là:

- Nghiên cứu thêm về mô hình DCNN để có thể tăng độ chính xác cho việc huấn luyện, nhận dạng hình ảnh từ đó ứng dụng trong thực tế.
- Có thể phát triển ứng dụng trên bộ dữ liệu đầy đủ và chi tiết trong nhiều lĩnh vực hơn.

TÀI LIỆU THAM KHẢO

1. Vũ Hữu Tiệp (2018), *Machine Learning cơ bản*, NXB Khoa học kỹ thuật, Hà Nội.
2. Nils J. Nilsson (1990), *Introduction to Machine Learning*
3. Trang hướng dẫn sử dụng PyTorch: <https://pytorch.org/tutorials/>
4. Convolutional Neural Networks (CNN) for CIFAR-10 Dataset, Parneet Kaur, <http://parneetk.github.io/blog/cnn-cifar10/>
5. PyTorch - Data loading, preprocess, display and torchvision, Jonathan Hui, https://jhui.github.io/2018/02/09/PyTorch-Data-loading-preprocess_torchvision/
6. Python torchvision.datasets.ImageFolder() Examples, nhiều tác giả, <https://www.programcreek.com/python/example/105102/torchvision.datasets.ImageFolder>