深入淺出 Python

2020.03.17 鄭余玄

@Deep Learning and Practice, Spring 2020

- python >= 3.6
  - `python -V`

Environments

- (NCTU) We'll provide workstation
- (Local) Install Python3
- (Cloud) Google Colab
- (Local / Remote) Jupyter Notebook

Format String

- 子串開頭有 f
- 字串內大括號包住變數

```
epoch, loss, acc = 600, 1.2345, 0.87654321

print(f'Epoch {epoch}, loss: {loss}, accuracy: {acc}')
# Epoch 600, loss: 1.2345, accuracy: 0.87654321

print(f'Epoch {epoch:4d}, loss: {loss:.2f}, accuracy: {acc:.2%}')
# Epoch  600, loss: 1.23, accuracy: 87.65%
```

format 方法

```
epoch, loss = 600, 0.12345

a = 'Epoch {epoch}, loss: {loss}'
print(a)
# Epoch {epoch}, loss: {loss}
print(a.format(epoch=epoch, loss=loss))
# Epoch 600, loss: 0.12345

b = 'Epoch {:4d}, loss: {:.2f}'
print(b)
# Epoch {:4d}, loss: {:.2f}
print(b.format(epoch, loss))
# Epoch  600, loss: 0.12
```

Built-in Functions

Ref link

enumerate

- enumerate(iterable, start=0)

```
data = [Img(), Img(), Img()]
for i, x in enumerate(data):
    print(i, x)
```

Example

```
data = [1, 3, 2]
print(max(data)) # 3
print(min(data)) # 1
print(sum(data)) # 6
print(len(data)) # 3

# Example: loss less than threshold
dones = [True, False, True, True]
all_done = all(dones) # False
any_done = any(dones) # True
```

open

```
# mode 'r': read (default)
# mode 'w': write
# mode 'a': append
f = open('test.txt', 'w')
f.write('zzz')
f.close()
```

```
with open('test.txt', 'w') as f:
    print('zzz', file=f)
    f.write('zzz')
```

zip

```
images = [Img(), Img(), Img()]
labels = [1, 1, 0]

for img, label in zip(images, labels):
    do_something(image, label)
```

```
for i, (img, label) in enumerate(zip(images, labels), start=1):
    do_something(image, label)
```

map

```
raw_data = ['1', '2', '3']
data = list(map(int, raw_data))
# [1, 2, 3]
```

```
raw_data = ['1', '2', '3']
data = list(map(lambda n : n + 'x', raw_data))
# ['1x', '2x', '3x']
```

Practice

```
channels = [32, 64, 128, 256, 512]

io_channel = [(32, 64), (64, 128), (128, 256), (256, 512)]

for in_channel, out_channel in io_channel:
    do_something(in_channel, out_channel)
```

Practice

- Hint: `zip`, slice

```
channels = [32, 64, 128, 256, 512]

io_channel = [(32, 64), (64, 128), (128, 256), (256, 512)]
```

防雷頁

Answer

```
channels = [32, 64, 128, 256, 512]

print(list(zip(channels[:-1], channels[1:])))

io_channel = [(32, 64), (64, 128), (128, 256), (256, 512)]
```

Review

```
r = zip([32, 64, 128, 256], [64, 128, 256, 512])
print(list(r))
# [(32, 64), (64, 128), (128, 256), (256, 512)]
```

```
channels = [32, 64, 128, 256, 512]
r = zip(channels[:-1], channels[1:])
print(list(r))
# [(32, 64), (64, 128), (128, 256), (256, 512)]
```

Example

```
channels = [32, 64, 128, 256, 512]

for in_channel, out_channel in zip(channels[:-1], channels[1:]):
    do_something(in_channel, out_channel)
```

Function

```
def f(x):
    return x ** 3 + 3 * (x ** 2) + 1

# Lambda function
f2 = lambda x : x ** 3 + 3 * (x ** 2) + 1

print(f(-1), f2(-1))
```

Positional Argument

```
def f(x=1, y=1):
    return x ** 3 + y ** 3

print(f(), f(0), f(0, 2))
print(f(y=3)) # 指定傳參數
```

```
# BatchNorm2d(num_features, eps=1e-05, momentum=0.1, affine=True
bn = BatchNorm2d(2)
bn = BatchNorm2d(num_features=2)
bn = BatchNorm2d(2, track_running_stats=False)
```

*args

```
def print_repeat(num, *content):
    for _ in range(num):
        print(content)

x = print_repeat(3, 'a', 'b', 'c')
```

Example

```
def p_norm(*v, p):
    ret = 0
    for k in v:
        ret += k ** p
    return ret ** (1 / p)

print(p_norm(1, 2, 3, p=2))
```

**kwargs

```
def create_residual_blocks(num, *args, **kwargs):
    return [nn.Conv2d(*args, **kwargs) for _ in range(num)]

x = create_residual_blocks(3, kernel_size=3, stride=1)
```

Iterable Unpacking

- `tuple`, `list`, `str`
- 底線 _ 忽略該變數

```
_, value = net(input)

dim = (4096, 3, 80, 80) # [N, C, H, W]
batch_size, _, height, width = dim
```

```
_, _, *size = dim
# size == [80, 80]

batch_size, *_ = dim
# batch_size == 4096
```

Example

```
def distance_square(p1, p2):
    (x1, y1), (x2, y2) = p1, p2
    dis = (x1 - x2) ** 2 + (y1 - y2) ** 2
    print(dis)

distance_square((1, 2), (3, 4))
```

Example

```
points = [(1, 2), (2, 2), (3, 4)]
for p in points:
    # p 是 tuple
    print(p[0], p[1])

# 結合 Unpacking
for x, y in points:
    # x 和 y 都是 int
    print(x, y)
```

Example

```
def norm_square(x, y):
    return x ** 2 + y ** 2

vector = (3, 4)

# method 1
print(norm_square(x=vector[0], y=vector[1]))

# method 2
x, y = vector
print(norm_square(x, y))

# method 3
print(norm_square(*vector))
```

Example

```
p1 = (1, 2)
p2 = (3, 4)

(x0, y0), (x1, y1) = p1, p2

x0, y0, (x1, y1) = *p1, p2

(x0, y0), x1, y1 = p1, *p2

x0, y0, x1, y1 = *p1, *p2
```

Dictionary Unpacking

```
# BatchNorm2d(num_features, eps=1e-05, momentum=0.1, affine=True
bn_args = {
    'momentum': None,
    'track_running_stats': False,
}
bn1 = BatchNorm2d(2, **bn_args)
bn2 = BatchNorm2d(25, **bn_args)
```

Practice: Transpose List of List

```
before = [
    [ 1,  2,  3,  4,  5],
    [ 6,  7,  8,  9, 10],
    [11, 12, 13, 14, 15],
]

after = [
  [1,  6, 11],
  [2,  7, 12],
  [3,  8, 13],
  [4,  9, 14],
  [5, 10, 15],
]
```

Practice: Transpose List of List

- One-liner Hint: `zip`, `list`, `map`

```
before = [
    [ 1,  2,  3,  4,  5],
    [ 6,  7,  8,  9, 10],
    [11, 12, 13, 14, 15],
]

after = [
  [1,  6, 11],
  [2,  7, 12],
  [3,  8, 13],
  [4,  9, 14],
  [5, 10, 15],
]
```

防雷頁

Answer: Transpose List of List

```
before = [
    [ 1,  2,  3,  4,  5],
    [ 6,  7,  8,  9, 10],
    [11, 12, 13, 14, 15],
]

print(list(map(list, zip(*before))))

after = [
  [1,  6, 11],
  [2,  7, 12],
  [3,  8, 13],
  [4,  9, 14],
  [5, 10, 15],
]
```

Review

```
r1 = zip([1, 2, 3, 4, 5], [6, 7, 8, 9, 10], [11, 12, 13, 14, 15]
print(list(r1))
# [(1, 6, 11), (2, 7, 12), (3, 8, 13), (4, 9, 14), (5, 10, 15)]

r2 = list(map(list, r1))
# [[1, 6, 11], [2, 7, 12], [3, 8, 13], [4, 9, 14], [5, 10, 15]]
```

```
r1 = zip(*before)
r2 = list(map(list, r1))

# in one line
result = list(map(list, zip(*before)))
```

List Comprehensions

```
# Original:
squares = []
for x in range(10):
    squares.append(x ** 2)


# List comprehension:
squares = [x ** 2 for x in range(10)]
```

Example

```
def preprocess(x):
    # do something
    return x

raw_data = ['This is demo.', 'For demo!']

input_data = [preprocess(x) for x in raw_data]
```

```
print([w.shape for w in weights])
```

```
data = [1, 2, 3]
data_tensor = [Tensor([x]) for x in data]
```

Example

```
raw_data = [(1, 2), (2, 2), (3, 4)]
data = [preprocess(x, y) for x, y in raw_data if x != y]
```

```
char_to_index = {'a': 0, 'b': 1}
index_to_char = {v: k for k, v in char_to_index.items()}
```

class

- inheritence from nn.Module

```
class MyNN(nn.Module):
  def __init__(self, input_size, hidden_size):
    super().__init__()
    self.layer = nn.Linear(input_size, hidden_size)

  def forward(self, x):
    return self.layer(x)
```

Example

```
class ReplayMemory:
  def __init__(self, capacity):
    self._buffer = deque(maxlen=capacity)

  def __len__(self):
    return len(self._buffer)

  def append(self, *transition):
    # (state, action, reward, next_state, done)
    self._buffer.append(tuple(map(tuple, transition)))

  def sample(self, batch_size=1):
    return random.sample(self._buffer, batch_size)
```

```
buffer = ReplayMemory(5000)
buffer.append(state, action, reward, next_state, done)
transitions = memory.sample(20)
```

Protocols

- Collection protocols
  - Sized: `__len__`
  - Container: `__contains__`
- Iteration protocols
  - Iterable: `__iter__`
  - Iterator: `__iter__`, `__next__`
- Context manager protocols
  - ContextManager: `__enter__`, `__exit__`

Python Standard Library

- `argparse`
- `collections`
  - `defaultdict`
  - `deque`
  - `Counter`
- `itertools`
  - `itertools.count`
  - `itertools.chain`

Useful Library

- `numpy`
- `matplotlib`
- `seaborn`
- NVIDIA DALI

Demo

- Google Colab
- Visual Studio Code (VSCode)
- CLI

Google Colab

- https://colab.research.google.com
- GPUs: 1xNvidia K80s, T4s, P4s and P100s
- VM lifetime: 12 hours

VSCode

CLI

- `python`
- `ipython`
- `python -c 'import torch'`
- `python -i test.py`