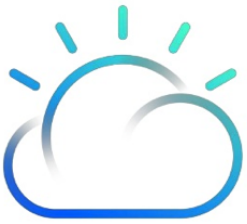


Watson Studio democratizes machine learning and deep learning to accelerate infusion of AI in your business to drive innovation. Watson Studio provides a suite of tools and a collaborative environment for data scientists, developers and domain experts.

(http://cocl.us/pytorch_link_top)

Simple Dataset

Table of Contents

In this lab, you will construct a basic dataset by using PyTorch and learn how to apply basic transformations to it.

Estimated Time Needed: **30 min**

Preparation

The following are the libraries we are going to use for this lab. The `torch.manual_seed()` is for forcing the random function to give the same number every time we try to recompile it.

In [1]:

```
# These are the libraries will be used for this lab.

import torch
from torch.utils.data import Dataset
torch.manual_seed(1)
```

Out[1]:

```
<torch._C.Generator at 0x7f1a840b1310>
```

Simple dataset

Let us try to create our own dataset class.

In [2]:

```
# Define class for dataset

class toy_set(Dataset):

    # Constructor with defult values
    def __init__(self, length = 100, transform = None):
        self.len = length
        self.x = 2 * torch.ones(length, 2)
        self.y = torch.ones(length, 1)
        self.transform = transform

    # Getter
    def __getitem__(self, index):
        sample = self.x[index], self.y[index]
        if self.transform:
            sample = self.transform(sample)
        return sample

    # Get Length
    def __len__(self):
        return self.len
```

Now, let us create our `toy_set` object, and find out the value on index 1 and the length of the inital dataset

In [3]:

```
# Create Dataset Object. Find out the value on index 1. Find out the length of Data set Object.

our_dataset = toy_set()
print("Our toy_set object: ", our_dataset)
print("Value on index 0 of our toy_set object: ", our_dataset[0])
print("Our toy_set length: ", len(our_dataset))
```

```
Our toy_set object:  <__main__.toy_set object at 0x7f1a112b92e8>
Value on index 0 of our toy_set object:  (tensor([2., 2.]), tensor
([1.]))
Our toy_set length:  100
```

As a result, we can apply the same indexing convention as a `list` , and apply the fuction `len` on the `toy_set` object. We are able to customize the indexing and length method by `def __getitem__(self, index)` and `def __len__(self)` .

Now, let us print out the first 3 elements and assign them to x and y:

In [4]:

```
# Use loop to print out first 3 elements in dataset

for i in range(3):
    x, y=our_dataset[i]
    print("index: ", i, '; x:', x, '; y:', y)
```

```
index:  0 ; x: tensor([2., 2.]) ; y: tensor([1.])
index:  1 ; x: tensor([2., 2.]) ; y: tensor([1.])
index:  2 ; x: tensor([2., 2.]) ; y: tensor([1.])
```

The dataset object is an Iterable; as a result, we apply the loop directly on the dataset object

In [5]:

```
for x,y in our_dataset:
    print(' x:', x, 'y:', y)
```

```
x: tensor([2., 2.]) y: tensor([1.])
x: tensor([2., 2.]) y: tensor([1.])
x: tensor([2., 2.]) y: tensor([1.])
x: tensor([2., 2.]) y: tensor([1.])
x: tensor([2., 2.]) y: tensor([1.])
x: tensor([2., 2.]) y: tensor([1.])
x: tensor([2., 2.]) y: tensor([1.])
x: tensor([2., 2.]) y: tensor([1.])
x: tensor([2., 2.]) y: tensor([1.])
x: tensor([2., 2.]) y: tensor([1.])
x: tensor([2., 2.]) y: tensor([1.])
x: tensor([2., 2.]) y: tensor([1.])
x: tensor([2., 2.]) y: tensor([1.])
x: tensor([2., 2.]) y: tensor([1.])
x: tensor([2., 2.]) y: tensor([1.])
x: tensor([2., 2.]) y: tensor([1.])
x: tensor([2., 2.]) y: tensor([1.])
x: tensor([2., 2.]) y: tensor([1.])
x: tensor([2., 2.]) y: tensor([1.])
x: tensor([2., 2.]) y: tensor([1.])
x: tensor([2., 2.]) y: tensor([1.])
x: tensor([2., 2.]) y: tensor([1.])
x: tensor([2., 2.]) y: tensor([1.])
x: tensor([2., 2.]) y: tensor([1.])
x: tensor([2., 2.]) y: tensor([1.])
x: tensor([2., 2.]) y: tensor([1.])
x: tensor([2., 2.]) y: tensor([1.])
x: tensor([2., 2.]) y: tensor([1.])
x: tensor([2., 2.]) y: tensor([1.])
x: tensor([2., 2.]) y: tensor([1.])
x: tensor([2., 2.]) y: tensor([1.])
x: tensor([2., 2.]) y: tensor([1.])
x: tensor([2., 2.]) y: tensor([1.])
x: tensor([2., 2.]) y: tensor([1.])
x: tensor([2., 2.]) y: tensor([1.])
x: tensor([2., 2.]) y: tensor([1.])
x: tensor([2., 2.]) y: tensor([1.])
x: tensor([2., 2.]) y: tensor([1.])
x: tensor([2., 2.]) y: tensor([1.])
x: tensor([2., 2.]) y: tensor([1.])
x: tensor([2., 2.]) y: tensor([1.])
x: tensor([2., 2.]) y: tensor([1.])
x: tensor([2., 2.]) y: tensor([1.])
x: tensor([2., 2.]) y: tensor([1.])
x: tensor([2., 2.]) y: tensor([1.])
x: tensor([2., 2.]) y: tensor([1.])
x: tensor([2., 2.]) y: tensor([1.])
x: tensor([2., 2.]) y: tensor([1.])
x: tensor([2., 2.]) y: tensor([1.])
x: tensor([2., 2.]) y: tensor([1.])
x: tensor([2., 2.]) y: tensor([1.])
x: tensor([2., 2.]) y: tensor([1.])
x: tensor([2., 2.]) y: tensor([1.])
x: tensor([2., 2.]) y: tensor([1.])
x: tensor([2., 2.]) y: tensor([1.])
x: tensor([2., 2.]) y: tensor([1.])
x: tensor([2., 2.]) y: tensor([1.])
```

```
x: tensor([2., 2.]) y: tensor([1.])
x: tensor([2., 2.]) y: tensor([1.])
x: tensor([2., 2.]) y: tensor([1.])
x: tensor([2., 2.]) y: tensor([1.])
x: tensor([2., 2.]) y: tensor([1.])
x: tensor([2., 2.]) y: tensor([1.])
x: tensor([2., 2.]) y: tensor([1.])
x: tensor([2., 2.]) y: tensor([1.])
x: tensor([2., 2.]) y: tensor([1.])
x: tensor([2., 2.]) y: tensor([1.])
x: tensor([2., 2.]) y: tensor([1.])
x: tensor([2., 2.]) y: tensor([1.])
x: tensor([2., 2.]) y: tensor([1.])
x: tensor([2., 2.]) y: tensor([1.])
x: tensor([2., 2.]) y: tensor([1.])
x: tensor([2., 2.]) y: tensor([1.])
x: tensor([2., 2.]) y: tensor([1.])
x: tensor([2., 2.]) y: tensor([1.])
x: tensor([2., 2.]) y: tensor([1.])
x: tensor([2., 2.]) y: tensor([1.])
x: tensor([2., 2.]) y: tensor([1.])
x: tensor([2., 2.]) y: tensor([1.])
x: tensor([2., 2.]) y: tensor([1.])
x: tensor([2., 2.]) y: tensor([1.])
x: tensor([2., 2.]) y: tensor([1.])
x: tensor([2., 2.]) y: tensor([1.])
x: tensor([2., 2.]) y: tensor([1.])
x: tensor([2., 2.]) y: tensor([1.])
x: tensor([2., 2.]) y: tensor([1.])
x: tensor([2., 2.]) y: tensor([1.])
x: tensor([2., 2.]) y: tensor([1.])
x: tensor([2., 2.]) y: tensor([1.])
x: tensor([2., 2.]) y: tensor([1.])
x: tensor([2., 2.]) y: tensor([1.])
x: tensor([2., 2.]) y: tensor([1.])
x: tensor([2., 2.]) y: tensor([1.])
x: tensor([2., 2.]) y: tensor([1.])
x: tensor([2., 2.]) y: tensor([1.])
x: tensor([2., 2.]) y: tensor([1.])
x: tensor([2., 2.]) y: tensor([1.])
x: tensor([2., 2.]) y: tensor([1.])
x: tensor([2., 2.]) y: tensor([1.])
x: tensor([2., 2.]) y: tensor([1.])
```

Practice

Try to create an `toy_set` object with length **50**. Print out the length of your object.

In [6]:

```
# Practice: Create a new object with length 50, and print the length of object out.
my_dataset = toy_set(length = 50)
print("My toy_set length: ", len(my_dataset))
```

```
My toy_set length:  50
```

Double-click **here** for the solution.

Transforms

You can also create a class for transforming the data. In this case, we will try to add 1 to x and multiply y by 2:

In [7]:

```
# Create tranform class add_mult

class add_mult(object):

    # Constructor
    def __init__(self, addx = 1, muly = 2):
        self.addx = addx
        self.muly = muly

    # Executor
    def __call__(self, sample):
        x = sample[0]
        y = sample[1]
        x = x + self.addx
        y = y * self.muly
        sample = x, y
        return sample
```

Now, create a transform object:.

In [8]:

```
# Create an add_mult transform object, and an toy_set object

a_m = add_mult()
data_set = toy_set()
```

Assign the outputs of the original dataset to `x` and `y`. Then, apply the transform `add_mult` to the dataset and output the values as `x_` and `y_`, respectively:

In [9]:

```
# Use loop to print out first 10 elements in dataset

for i in range(10):
    x, y = data_set[i]
    print('Index: ', i, 'Original x: ', x, 'Original y: ', y)
    x_, y_ = a_m(data_set[i])
    print('Index: ', i, 'Transformed x_:', x_, 'Transformed y_:', y_)
```

```
Index:  0 Original x:  tensor([2., 2.]) Original y:  tensor([1.])
Index:  0 Transformed x_: tensor([3., 3.]) Transformed y_: tensor([2.])
Index:  1 Original x:  tensor([2., 2.]) Original y:  tensor([1.])
Index:  1 Transformed x_: tensor([3., 3.]) Transformed y_: tensor([2.])
Index:  2 Original x:  tensor([2., 2.]) Original y:  tensor([1.])
Index:  2 Transformed x_: tensor([3., 3.]) Transformed y_: tensor([2.])
Index:  3 Original x:  tensor([2., 2.]) Original y:  tensor([1.])
Index:  3 Transformed x_: tensor([3., 3.]) Transformed y_: tensor([2.])
Index:  4 Original x:  tensor([2., 2.]) Original y:  tensor([1.])
Index:  4 Transformed x_: tensor([3., 3.]) Transformed y_: tensor([2.])
Index:  5 Original x:  tensor([2., 2.]) Original y:  tensor([1.])
Index:  5 Transformed x_: tensor([3., 3.]) Transformed y_: tensor([2.])
Index:  6 Original x:  tensor([2., 2.]) Original y:  tensor([1.])
Index:  6 Transformed x_: tensor([3., 3.]) Transformed y_: tensor([2.])
Index:  7 Original x:  tensor([2., 2.]) Original y:  tensor([1.])
Index:  7 Transformed x_: tensor([3., 3.]) Transformed y_: tensor([2.])
Index:  8 Original x:  tensor([2., 2.]) Original y:  tensor([1.])
Index:  8 Transformed x_: tensor([3., 3.]) Transformed y_: tensor([2.])
Index:  9 Original x:  tensor([2., 2.]) Original y:  tensor([1.])
Index:  9 Transformed x_: tensor([3., 3.]) Transformed y_: tensor([2.])
```

As the result, `x` has been added by 1 and y has been multiplied by 2, as *[2, 2] + 1 = [3, 3]* and *[1] x 2 = [2]*

We can apply the transform object every time we create a new `toy_set object` ? Remember, we have the constructor in toy_set class with the parameter `transform = None` . When we create a new object using the constructor, we can assign the transform object to the parameter transform, as the following code demonstrates.

In [10]:

```
# Create a new data_set object with add_mult object as transform

cust_data_set = toy_set(transform = a_m)
```

This applied `a_m` object (a transform method) to every element in `cust_data_set` as initialized. Let us print out the first 10 elements in `cust_data_set` in order to see whether the `a_m` applied on `cust_data_set`

In [11]:

```
# Use loop to print out first 10 elements in dataset

for i in range(10):
    x, y = data_set[i]
    print('Index: ', i, 'Original x: ', x, 'Original y: ', y)
    x_, y_ = cust_data_set[i]
    print('Index: ', i, 'Transformed x_:', x_, 'Transformed y_:', y_)
```

```
Index:  0 Original x:  tensor([2., 2.]) Original y:  tensor([1.])
Index:  0 Transformed x_: tensor([3., 3.]) Transformed y_: tensor([2.])
Index:  1 Original x:  tensor([2., 2.]) Original y:  tensor([1.])
Index:  1 Transformed x_: tensor([3., 3.]) Transformed y_: tensor([2.])
Index:  2 Original x:  tensor([2., 2.]) Original y:  tensor([1.])
Index:  2 Transformed x_: tensor([3., 3.]) Transformed y_: tensor([2.])
Index:  3 Original x:  tensor([2., 2.]) Original y:  tensor([1.])
Index:  3 Transformed x_: tensor([3., 3.]) Transformed y_: tensor([2.])
Index:  4 Original x:  tensor([2., 2.]) Original y:  tensor([1.])
Index:  4 Transformed x_: tensor([3., 3.]) Transformed y_: tensor([2.])
Index:  5 Original x:  tensor([2., 2.]) Original y:  tensor([1.])
Index:  5 Transformed x_: tensor([3., 3.]) Transformed y_: tensor([2.])
Index:  6 Original x:  tensor([2., 2.]) Original y:  tensor([1.])
Index:  6 Transformed x_: tensor([3., 3.]) Transformed y_: tensor([2.])
Index:  7 Original x:  tensor([2., 2.]) Original y:  tensor([1.])
Index:  7 Transformed x_: tensor([3., 3.]) Transformed y_: tensor([2.])
Index:  8 Original x:  tensor([2., 2.]) Original y:  tensor([1.])
Index:  8 Transformed x_: tensor([3., 3.]) Transformed y_: tensor([2.])
Index:  9 Original x:  tensor([2., 2.]) Original y:  tensor([1.])
Index:  9 Transformed x_: tensor([3., 3.]) Transformed y_: tensor([2.])
```

The result is the same as the previous method.

In [12]:

```
# Practice: Construct your own my_add_mult transform. Apply my_add_mult on a new to
y_set object. Print out the first three elements from the transformed dataset.
my_add_mult = toy_set(transform = a_m)
for i in range (4):
    x, y = data_set[i]
    print('Index: ', i, 'Original x: ', x, 'Original y: ', y)
    x_, y_ = my_add_mult[i]
    print('Index: ', i, 'Transformed x_: ', x_, 'Transformed y_: ', y_)
```

```
Index:  0 Original x:  tensor([2., 2.]) Original y:  tensor([1.])
Index:  0 Transformed x_:  tensor([3., 3.]) Transformed y_:  tensor
([2.])
Index:  1 Original x:  tensor([2., 2.]) Original y:  tensor([1.])
Index:  1 Transformed x_:  tensor([3., 3.]) Transformed y_:  tensor
([2.])
Index:  2 Original x:  tensor([2., 2.]) Original y:  tensor([1.])
Index:  2 Transformed x_:  tensor([3., 3.]) Transformed y_:  tensor
([2.])
Index:  3 Original x:  tensor([2., 2.]) Original y:  tensor([1.])
Index:  3 Transformed x_:  tensor([3., 3.]) Transformed y_:  tensor
([2.])
```

In [13]:

```
class my_add_mult(object):
    def __init__(self, add = 2, mul = 10):
        self.add=add
        self.mul=mul

    def __call__(self, sample):
        x = sample[0]
        y = sample[1]
        x = x + self.add
        y = y + self.add
        x = x * self.mul
        y = y * self.mul
        sample = x, y
        return sample


my_dataset = toy_set(transform = my_add_mult())
for i in range(3):
    x_, y_ = my_dataset[i]
    print('Index: ', i, 'Transformed x_:', x_, 'Transformed y_:', y_)
```

```
Index:  0 Transformed x_: tensor([40., 40.]) Transformed y_: tensor([3
0.])
Index:  1 Transformed x_: tensor([40., 40.]) Transformed y_: tensor([3
0.])
Index:  2 Transformed x_: tensor([40., 40.]) Transformed y_: tensor([3
0.])
```

Double-click **here** for the solution. <!-- class my_add_mult(object):
def **init**(self, add = 2, mul = 10): self.add=add self.mul=mul

```
    def __call__(self, sample):
        x = sample[0]
        y = sample[1]
        x = x + self.add
        y = y + self.add
        x = x * self.mul
        y = y * self.mul
        sample = x, y
        return sample
```

my_dataset = toy_set(transform = my_add*mult()) for i in range(3): x*, y_ = my*dataset[i] print('Index: ', i,* '*Transformed x*:', x, '*Transformed y*:', y_)

-->

Compose

You can compose multiple transforms on the dataset object. First, import `transforms` from `torchvision` :

In [14]:

```
# Run the command below when you do not have torchvision installed
# !conda install -y torchvision

from torchvision import transforms
```

Then, create a new transform class that multiplies each of the elements by 100:

In [15]:

```
# Create tranform class mult

class mult(object):

    # Constructor
    def __init__(self, mult = 100):
        self.mult = mult

    # Executor
    def __call__(self, sample):
        x = sample[0]
        y = sample[1]
        x = x * self.mult
        y = y * self.mult
        sample = x, y
        return sample
```

Now let us try to combine the transforms `add_mult` and `mult`

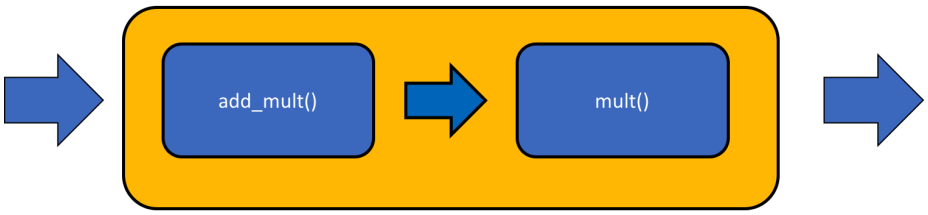
In [16]:

```
# Combine the add_mult() and mult()

data_transform = transforms.Compose([add_mult(), mult()])
print("The combination of transforms (Compose): ", data_transform)
```

```
The combination of transforms (Compose):  Compose(
    <__main__.add_mult object at 0x7f1a10379a20>
    <__main__.mult object at 0x7f1a10379a58>
)
```

The new `Compose` object will perform each transform concurrently as shown in this figure:



In [17]:

```
data_transform(data_set[0])
```

Out[17]:

```
(tensor([300., 300.]), tensor([200.]))
```

In [18]:

```
x,y=data_set[0]
x_,y_=data_transform(data_set[0])
print( 'Original x: ', x, 'Original y: ', y)

print( 'Transformed x_:', x_, 'Transformed y_:', y_)
```

```
Original x:  tensor([2., 2.]) Original y:  tensor([1.])
Transformed x_: tensor([300., 300.]) Transformed y_: tensor([200.])
```

Now we can pass the new `Compose` object (The combination of methods `add_mult()` and `mult`) to the constructor for creating `toy_set` object.

In [19]:

```
# Create a new toy_set object with compose object as transform

compose_data_set = toy_set(transform = data_transform)
```

Let us print out the first 3 elements in different `toy_set` datasets in order to compare the output after different transforms have been applied:

In [20]:

```
# Use loop to print out first 3 elements in dataset

for i in range(3):
    x, y = data_set[i]
    print('Index: ', i, 'Original x: ', x, 'Original y: ', y)
    x_, y_ = cust_data_set[i]
    print('Index: ', i, 'Transformed x_:', x_, 'Transformed y_:', y_)
    x_co, y_co = compose_data_set[i]
    print('Index: ', i, 'Compose Transformed x_co: ', x_co ,'Compose Transformed y_
co: ',y_co)
```

```
Index:  0 Original x:  tensor([2., 2.]) Original y:  tensor([1.])
Index:  0 Transformed x_: tensor([3., 3.]) Transformed y_: tensor([2.])
Index:  0 Compose Transformed x_co:  tensor([300., 300.]) Compose Trans
formed y_co:  tensor([200.])
Index:  1 Original x:  tensor([2., 2.]) Original y:  tensor([1.])
Index:  1 Transformed x_: tensor([3., 3.]) Transformed y_: tensor([2.])
Index:  1 Compose Transformed x_co:  tensor([300., 300.]) Compose Trans
formed y_co:  tensor([200.])
Index:  2 Original x:  tensor([2., 2.]) Original y:  tensor([1.])
Index:  2 Transformed x_: tensor([3., 3.]) Transformed y_: tensor([2.])
Index:  2 Compose Transformed x_co:  tensor([300., 300.]) Compose Trans
formed y_co:  tensor([200.])
```

Let us see what happened on index 0. The original value of `x` is *[2, 2]*, and the original value of `y` is [1]. If we only applied `add_mult()` on the original dataset, then the `x` became *[3, 3]* and y became *[2]*. Now let us see what is the value after applied both `add_mult()` and `mult()`. The result of x is *[300, 300]* and y is *[200]*. The calculation which is equavalent to the compose is *x = ([2, 2] + 1) x 100 = [300, 300], y = ([1] x 2) x 100 = 200*

Practice

Try to combine the `mult()` and `add_mult()` as `mult()` to be executed first. And apply this on a new `toy_set` dataset. Print out the first 3 elements in the transformed dataset.

In [23]:

```
# Practice: Make a compose as mult() execute first and then add_mult(). Apply the c
ompose on toy_set dataset. Print out the first 3 elements in the transformed datase
t.
my_compose = transforms.Compose([mult(), add_mult()])
my_transformed_dataset = toy_set(transform = my_compose)
for i in range(3):
    x_, y_ = my_transformed_dataset[i]
    print('Index: ', i, 'Transformed x_: ', x_, 'Transformed y_: ', y_)
```

```
Index:  0 Transformed x_:  tensor([201., 201.]) Transformed y_:  tensor
([200.])
Index:  1 Transformed x_:  tensor([201., 201.]) Transformed y_:  tensor
([200.])
Index:  2 Transformed x_:  tensor([201., 201.]) Transformed y_:  tensor
([200.])
```

Double-click **here** for the solution.

About the Authors:

Joseph Santarcangelo (https://www.linkedin.com/in/joseph-s-50398b136/) has a PhD in Electrical Engineering, his research focused on using machine learning, signal processing, and computer vision to determine how videos impact human cognition. Joseph has been working for IBM since he completed his PhD.

Other contributors: Michelle Carey (https://www.linkedin.com/in/michelleccarey/), Mavis Zhou (www.linkedin.com/in/jiahui-mavis-zhou-a4537814a)
