zero-base /

Life Changing Education

Chapter 48. pytorch를 이용한 mnist 데이터 분류

이번에는 MNIST로 torch 연습

일단 import

5초

```
1 import torch
2 import torch.nn as nn
3 import torch.nn.functional as F
4 import torch.optim as optim
5 from torchvision import datasets, transforms
6
7 from matplotlib import pyplot as plt
8 %matplotlib inline
```

Cuda가 가능하면 cuda 아니면 cpu 설정

0초

```
1 is_cuda = torch.cuda.is_available()
2 device = torch.device('cuda' if is_cuda else 'cpu')
3
4 print ('Current cuda device is', device)
```

```
Current cuda device is cuda
```

zero-base /

파라미터 설정

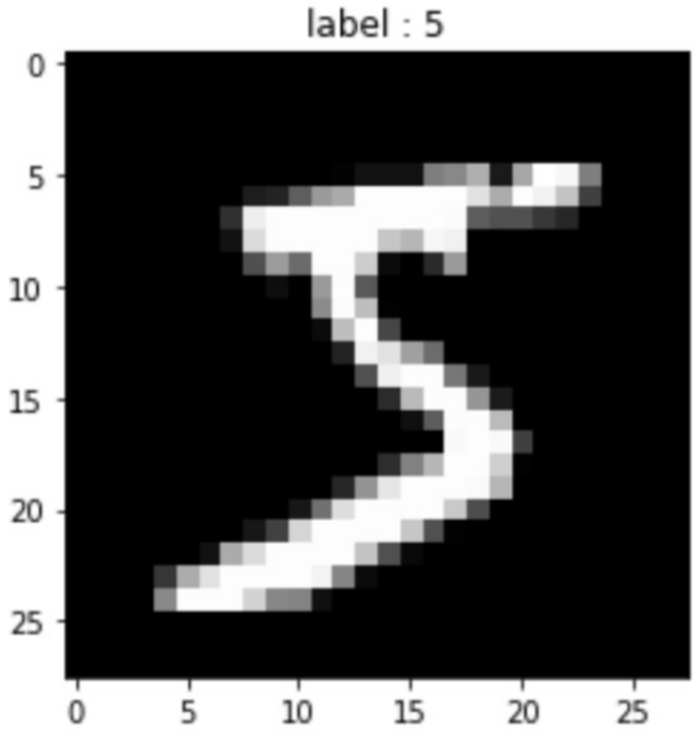
MNIST 데이터 불러오기

```
train_data = datasets.MNIST(root = './data',
                            train = True,
                            download = True,
                            transform = transforms.ToTensor())
test_data = datasets.MNIST(root = './data',
                           train = False,
                           transform = transforms.ToTensor())

print('number of training data: ', len(train_data))
print('number of test data: ', len(test_data))
```

[5] 0초

```
1 image, label = train_data[0]
2
3 plt.imshow(image.squeeze().numpy(), cmap = 'gray')
4 plt.title('label : %s' % label)
5 plt.show()
```



미니 배치 구성

```
train_loader = torch.utils.data.DataLoader(dataset = train_data,
                                           batch_size = batch_size, shuffle = True)
test_loader  = torch.utils.data.DataLoader(dataset = test_data,
                                           batch_size = batch_size, shuffle = True)

first_batch = train_loader.__iter__().__next__()
```

```
print('{:15s} | {:<25s} | {}'.format('name', 'type', 'size'))
print('{:15s} | {:<25s} | {}'.format('Num of Batch', '', len(train_loader)))
print('{:15s} | {:<25s} | {}'.format('first_batch', str(type(first_batch)),
                                     len(first_batch)))
print('{:15s} | {:<25s} | {}'.format('first_batch[0]', str(type(first_batch[0])),
                                     first_batch[0].shape))
print('{:15s} | {:<25s} | {}'.format('first_batch[1]', str(type(first_batch[1])),
                                     first_batch[1].shape))
```

```
name            | type                      | size
Num of Batch    |                           | 1200
first_batch     | <class 'list'>            | 2
first_batch[0]  | <class 'torch.Tensor'>    | torch.Size([50, 1, 28, 28])
first_batch[1]  | <class 'torch.Tensor'>    | torch.Size([50])
```

```python
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, 3, 1, padding='same')
        self.conv2 = nn.Conv2d(32, 64, 3, 1, padding='same')
        self.dropout = nn.Dropout2d(0.25)
        self.fc1 = nn.Linear(3136, 1000) #7*7*64 = 3136
        self.fc2 = nn.Linear(1000, 10)

    def forward(self, x):
        x = self.conv1(x)
        x = F.relu(x)
        x = F.max_pool2d(x, 2)
        x = self.conv2(x)
        x = F.relu(x)
        x = F.max_pool2d(x, 2)
        x = self.dropout(x)
        x = torch.flatten(x, 1)
        x = self.fc1(x)
        x = F.relu(x)
        x = self.fc2(x)
        output = F.log_softmax(x, dim=1)
        return output
```

```
model = CNN().to(device)
optimizer = optim.Adam(model.parameters(), lr = learning_rate)
criterion = nn.CrossEntropyLoss( )
```

```python
model.train()
i = 1
for epoch in range(epoch_num):
    for data, target in train_loader:
        data = data.to(device)
        target = target.to(device)
        optimizer.zero_grad()
        output = model(data)
        loss = criterion(output, target)
        loss.backward()
        optimizer.step()
        if i % 1000 == 0:
            print('Train Step: {}\tLoss: {:.3f}'.format(i, loss.item()))
        i += 1
```

```
Train Step: 1000        Loss: 0.186
Train Step: 2000        Loss: 0.188
```

```
model.eval()
correct = 0
for data, target in test_loader:
#     data, target = Variable(data, volatile=True), Variable(target)
    data = data.to(device)
    target = target.to(device)
    output = model(data)
    prediction = output.data.max(1)[1]
    correct += prediction.eq(target.data).sum()

print('Test set: Accuracy: {:.2f}%'.format(100. * correct / len(test_loader.dataset)))
```

```
Test set: Accuracy: 99.16%
```