exam 개인학습

Lec 10

Type something...

01) MNIST Introduction

01) MNIST Introduction

Reading data

```
import torchvision.datasets as dsets
...
mnist_train = dsets.MNIST(root="MNIST_data/", train=True, transform=transforms.ToTensor(),
download=True)

mnist_test = dsets.MNIST(root="MNIST_data/", train=False, transform=transforms.ToTensor(),
download=True)

data_loader = torch.utils.DataLoader(DataLoader=mnist_train, batch_size=batch_size,
shuffle=True, drop_last=True)
...
for epoch in range(training_epochs):
...
    for X, Y in data_loader:
        # reshape input image into [batch_size by 784]
        # label is not one-hot encoded
        X = X.view(-1, 28 * 28).to(device)
```

`mnist_train = dsets.MNIST(root = "MNIST_data/", train = True, transform = transforms.ToTensor(), download = True)` : train set

`mnist_test = dsets.MNIST(root = "MNIST_data/", train = False, transform = transform.ToTensor(), download = True)` :test set

`data_loader = torch.utils.DataLoader(DataLoader = mnist_train, batch_size = batch_size, shuffle = True, drop_last = True)` :dataloader를 통해 minibatch학습 진행

- softmax를 이용하여 학습을 진행

Softmax

```
# MNIST data image of shape 28 * 28 = 784
linear = torch.nn.Linear(784, 10, bias=True).to(device)
# initialization
torch.nn.init.normal_(linear.weight)
# parameters
training_epochs = 15
batch_size = 100
# define cost/Loss & optimizer
criterion = torch.nn.CrossEntropyLoss().to(device)    # Softmax is internally
optimizer = torch.optim.SGD(linear.parameters(), lr=0.1)

for epoch in range(training_epochs):
    avg_cost = 0
    total_batch = len(data_loader)
    for X, Y in data_loader:
        # reshape input image into [batch_size by 784]
        # label is not one-hot encoded
        X = X.view(-1, 28 * 28).to(device)
        optimier.zero_grad()
        hypothesis = linear(X)
        cost = criterion(hypothesis, Y)
        cost.backward()
        avg_cost += cost / total_batch
    print("Epoch: ", "%04d" % (epoch+1), "cost =", "{:.9f}".format(avg_cost))
```

```
Epoch: 0001 cost = 2.511683702
Epoch: 0002 cost = 0.977319956
Epoch: 0003 cost = 0.797017217
Epoch: 0004 cost = 0.710427940
Epoch: 0005 cost = 0.655205429
Epoch: 0006 cost = 0.615207732
Epoch: 0007 cost = 0.584421575
Epoch: 0008 cost = 0.559486568
Epoch: 0009 cost = 0.538655698
Epoch: 0010 cost = 0.520880997
Epoch: 0011 cost = 0.505315244
Epoch: 0012 cost = 0.491431117
Epoch: 0013 cost = 0.479477882
Epoch: 0014 cost = 0.468681127
Epoch: 0015 cost = 0.458788306
Learning finished
Accuracy: 0.8718999624252319
```

`linear = torch.nn.Linear(28*28, 10, bias = True).to(device)` #784 * 10 Linear, 0~9분류

`torch.nn.init.normal_(linear.weight)` # 초기화

`training_epochs = 15`

`batch_size = 100`

`criterion = torch.nn.CrossEntropyLoss().to(device)`

`optimizer = torch.optim.SGD(linear.parameters(), lr = 0.1` # SGD GD

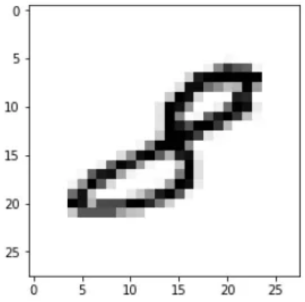
```
for epoch in range(training_epochs):
    avg_cost = 0
    total_batch = len(data_loader)
    for X, Y in data_loader:
        X = X.view(-1, 28 * 28).to(device) # batch_size * 756
        hypothesis = linear(X) # Linear
        cost = criterion(hypothesiss, Y) # softmax cost
        optimizer.zero_grad()
        cost.backward()
        optimizer.step()
        avg_cost += cost / total_batch # batch마다 cost를 더해주고 total_batch를 나눠주기
    print("Epoch: ", "%04d" %(epoch+1), "cost = ", "{:.9f}".format(avg_cost))
```

-

In [9]:

```
# Test the model using test sets
with torch.no_grad():
    X_test = mnist_test.test_data.view(-1, 28 * 28).float().to(device)
    Y_test = mnist_test.test_labels.to(device)

    prediction = linear(X_test)
    correct_prediction = torch.argmax(prediction, 1) == Y_test
    accuracy = correct_prediction.float().mean()
    print('Accuracy:', accuracy.item())

    # Get one and predict
    r = random.randint(0, len(mnist_test) - 1)
    X_single_data = mnist_test.test_data[r:r + 1].view(-1, 28 * 28).float().to(device)
    Y_single_data = mnist_test.test_labels[r:r + 1].to(device)

    print('Label: ', Y_single_data.item())
    single_prediction = linear(X_single_data)
    print('Prediction: ', torch.argmax(single_prediction, 1).item())

    plt.imshow(mnist_test.test_data[r:r + 1].view(28, 28), cmap='Greys', interpolation='nearest')
    plt.show()
```

```
Accuracy: 0.8862999677658081
Label:  8
Prediction:  3
```




```
with torch.no_grad():
    X_test = mnist_test.test_data.view(-1, 28*28).float().to(device)
    Y_test = mnist_test.test_labels.to(device)

    prediction = linear(X_test)
    corrct_prediction = torch.argmax(prediction, dim = 1) == Y_test # 열로 이동하기 때문에 dim = 1 → 각 행의 최대값의 인덱스를 뽑아준다. 최대값은 1.
    accuracy = correct_prediction.float().mean()
    print('Accruacy: accuracy.item()) # torch에서 값을 꺼낼 때는 item() 사용

    r = random.randint(0, len(mnist_test) - 1)
    X_single_data = mnist_test.test_data[r:r + 1].view(-1, 28 * 28).float().to(device) # 1개의 데이터 꺼내서 비교해보기
    Y_single_data = mnist_test.test_labels[r:r+1].to(device)
    print('Label: ', Y_single_data.item())
    single_prediction = linear(X_single_data)
    print('Prediction: ', torch.argmax(single_prediction, dim = 1).item())

    plt.imshow(mnist_test.test_data[r:r+1].view(28, 28) ,cmap = 'Greys', interpolation='nearest')
    plt.show()
```

- Linear를 1개만 쓰는 모델의 성능은 88%로 좋다고 말하기 애매한 수준이다.