합성곱신경망(CNN)을 이용한 손글씨 인식

문서 정보

- 작성일 : 2023. 06. 23
- 작성자 : 조성민
- 학번 : B877034
- 과목 : 인공지능응용
- Git : https://github.com/josungmin/AI-Final-Exam

인공지능응용 강의는 합성곱신경망(CNN, Convolution Neural Network)을 중심으로 진행되었다. 실습으로 경험한 'Cifar10' 데이터셋을 이용한 이미지 분류기와 비견되는 'MNIST' 데이터셋을 이용한 손 글씨 인식을 과제로 진행하였다.

1. 시드 값 고정

```
# 랜덤 시드 고정
torch.manual_seed(777)

if device == 'cuda':
    torch.cuda.manual_seed_all(777)
```

학습을 위한 실험 시 무작위성을 컨트롤하기 위해서 시드를 고정한다고 한다. 만약 무작위로 지정된 시드의 값을 사용한 학습 모델 결과에 문제가 생긴 경우를 가정해 본다면, 무작위로 정해지는 다음 시드 값으로 학습된 모델도 같은 문제가 발생한다고 단언할 수 없다. 때문에 무작위성을 배제하고 재현할 수 있는(일관성 있는) 학습결과를 얻기 위해 시드를 고정한다고 한다.

2. 학습에 사용할 파라미터 설정

```
learning_rate = 0.001
training_epochs = 15
batch_size = 100
```

Learning Rate 는 0.001, 전체 데이터 셋에 대한 학습 횟수는 15 회, 미니 배치의 크기는 100 으로 설정하였다.

3. 데이터셋 정의

```
mnist_train = dsets.MNIST(root='MNIST_data/',   # 다운로드 경로 지정
                          train=True,           # True를 지정하면 훈련 데이터로 다운로드
                          transform=transforms.ToTensor(), # 텐서로 변환
                          download=True)

mnist_test = dsets.MNIST(root='MNIST_data/',    # 다운로드 경로 지정
                         train=False,           # False를 지정하면 테스트 데이터로 다운로드
                         transform=transforms.ToTensor(), # 텐서로 변환
                         download=True)
```

'MNIST' 데이터셋은 총 70000 개의 이미지 중 60,000 개를 학습용으로, 10,000 개를 테스트용으로 미리 구분해 놓았다. 'MNIST_data'폴더 하위에 Tensor 로 변환하여 각각 저장한다.

4. 데이터 로더를 사용하여 배치 크기를 지정

```
data_loader = torch.utils.data.DataLoader(dataset=mnist_train,
                                          batch_size=batch_size,
                                          shuffle=True,
                                          drop_last=True)
```

전체 데이터를 하나의 행렬로 선언하여 전체 데이터의 대한 경사 하강법을 수행하는 것은 데이터의 양이 많아 질수록 느려지고, 계산량이 많아진다. 때문에 전체 데이터를 더 작은 단위로 나누어 행당 단위로 학습하는 개념인 미니배치를 적용하여 해결한다고 한다. 해당 코드에서는 전체 데이터(60,000 개)를 100 개의 단위로 나누어 작업을 수행한다.

5. 합성곱신경망 모델 설계 및 정의

```
class CNN(torch.nn.Module):

    def __init__(self):
        super(CNN, self).__init__()

        self.layer1 = torch.nn.Sequential(
            torch.nn.Conv2d(1, 32, kernel_size=3, stride=1, padding=1),
            torch.nn.ReLU(),
            torch.nn.MaxPool2d(kernel_size=2, stride=2))

        self.layer2 = torch.nn.Sequential(
            torch.nn.Conv2d(32, 64, kernel_size=3, stride=1, padding=1),
            torch.nn.ReLU(),
            torch.nn.MaxPool2d(kernel_size=2, stride=2))

        # 전결합층 7x7x64 inputs -> 10 outputs
        self.fc = torch.nn.Linear(7 * 7 * 64, 10, bias=True)

        # 전결합층 한정으로 가중치 초기화
        torch.nn.init.xavier_uniform_(self.fc.weight)

    def forward(self, x):
        out = self.layer1(x)
        out = self.layer2(out)
        out = out.view(out.size(0), -1)   # 전결합층을 위해서 Flatten
        out = self.fc(out)
        return out

# CNN 모델 정의
model = CNN().to(device)
```

5. Loss 함수와 Optimizer 정의

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

Pytorch 가 제공하는 'torch.nn.CrossEntropyLoss()' Loss 함수를 사용하였으며, 이는 다중 분류를 위한 대표적인 Loss 함수이다. Optimizer 또한 Pytorch 가 제공하는 아담(Adam)을 사용하였다.

6. 학습 진행

```
for epoch in range(training_epochs):
    running_loss  = 0

    for X, Y in data_loader: # 미니 배치 단위로 꺼내온다. X: 미니 배치, Y: 는레이블.
        X = X.to(device)
        Y = Y.to(device)

        optimizer.zero_grad()
        hypothesis = model(X)
        loss = criterion(hypothesis, Y)
        loss.backward()
        optimizer.step()

        running_loss  += loss / total_batch

    print('[Epoch: {:>4}] cost = {:>.9}'.format(epoch + 1, running_loss ))
```

테스트 이미지에 대해 모델을 통과시켜서 손실을 계산한다. 학습이 한번 완료된 시점에 running Loss 출력함으로써 학습이 잘 진행되고 있는지 확인한다.

7. 테스트

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

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

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

    print('Lable: ' , Y_single_data.item())
    single_prediction = model(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()
```

테스트 데이터 중 무작위 이미지를 선택하여 학습이 정확도를 파악한다.