대규모 데이터셋 처리

거대한 데이터를 하나의 ndarray 인스턴스로 처리하면 모든 원소를 한꺼번에 메모리에 올려야 하는데, 이러한 대규모 데이터를 처리할 수 있도록 데이터셋 전용으로 Dataset 클래스를 만든다.
Dataset 클래스에 데이터를 전처리할 수 있는 구조도 추가해준다.

큰 데이터셋 처리

BigData 클래스를 초기화할 때는 데이터를 읽지 않고 __getitem__(index)가 불리는 시점에 데이터를 읽는다.

데이터 이어 붙이기

신경망 입력 형태로의 데이터를 준비하기 위해서 데이터 셋 중 일부를 미니배치로 꺼낸다.
인덱스를 지정하여 batch에 여러 데이터를 리스트로 저장하고, ndarray 인스턴스로 변환단다.

```
train_set = ds.Spiral(train = True)

batch_index = [0, 1, 2]
batch = [train_set[i] for i in batch_index]

x = np.array([example[0] for example in batch])
t = np.array([example[1] for example in batch])

print(x.shape)
print(t.shape)
```

```
(3, 2)
(3,)
```

batch의 각 원소에서 데이터만 꺼내 하나의 ndarray 인스턴스로 변형한다.

미니배치를 만드는 부분을 수정하여 훨씬 큰 데이터셋을 대응할 수 있다.
데이터셋 인터페이스를 통일하여 다양한 데이터셋을 똑같은 코드로 처리할 수 있다.

데이터셋 전처리

모델에 데이터를 입력하기 전에 데이터를 가공해야 한다.
데이터 형상 변형, 데이터 확장 등 다양한 방법이 있다.
transform은 입력 데이터 하나에 대한 변환을 처리하고, target_transform 레이블 하나에 대한 변환을 처리한다.(데이터셋에 대해 사용자가 원하는 전처리를 추가할 수 있음)

미니배치를 뽑아주는 DataLoader

반복자란 원소를 반복하여 꺼내주는 역할을 한다.
반복자 구조를 이용하여 미니배치를 뽑아준다. 주어진 데이터셋의 첫 데이터부터 차례로 꺼내주지만, 필요에 따라 뒤섞일 수 있다.
DataLoader 클래스를 사용하면 미니배치를 간단히 뽑아준다.
용도(train, test)에 맞게 두 개의 DataLoader를 생성한다.
train DataLoader는 epoch별로 데이터를 뒤섞어야 하기 때문에 shuffle = True로 설정한다.
미니배치 추출과 데이터 뒤섞기는 DataLoader가 알아서 해준다.

```
from dezero.datasets import Spiral
from dezero.dataloaders import DataLoader

batch_size = 10
max_epoch = 1

train_set = Spiral(train = True)
test_set = Spiral(train = False)
train_loader = DataLoader(train_set, batch_size)
test_loader = DataLoader(test_set, batch_size, shuffle = False)

for epoch in range(max_epoch):
    for x, t in train_loader:
        print(x.shape, t.shape)
        break

    for x, t in test_loader:
        print(x.shape, t.shape)
        break
```

```
(10, 2) (10,)
(10, 2) (10,)
```

accuracy 함수는 인식 정확도를 평가해주는 함수이다.
인수 y(신경망의 예측 결과)와 t(정답 데이터)를 받아서 정답률을 계산해준다.
신경망의 예측 결과를 pred에 저장하고 최대값을 찾아서 항상 변경한다.

```
y = np.array([[0.2, 0.8, 0], [0.1, 0.9, 0], [0.8, 0.1, 0.1]])
t = np.array([1, 2, 0])
acc = F.accuracy(y, t)
print(acc)
```

```
variable(0.6666666666666666)
```

스파이럴 데이터셋 학습

1. DataLoader를 사용해서 미니배치를 추출한다.
2. accuracy 함수를 사용하여 인식 정확도를 계산한다.
3. epoch별로 test 데이터셋을 사용하여 훈련 결과를 평가한다. - test는 역전파가 필요없다.
4. test DataLoader에서 미니배치 데이터를 꺼내 평가한다.
5. accuracy 함수를 사용하여 인식 정확도를 계산한다.

손실과 인식 정확도 추이

epoch가 진행됨에 따라 loss가 낮아지고 accuracy는 상승한다.
학습이 제대로 이루어지고 있음을 나타낸다.
train과 test 사이에 과적합이 있다고 볼 수 없다는 뜻이다.

MNIST 학습

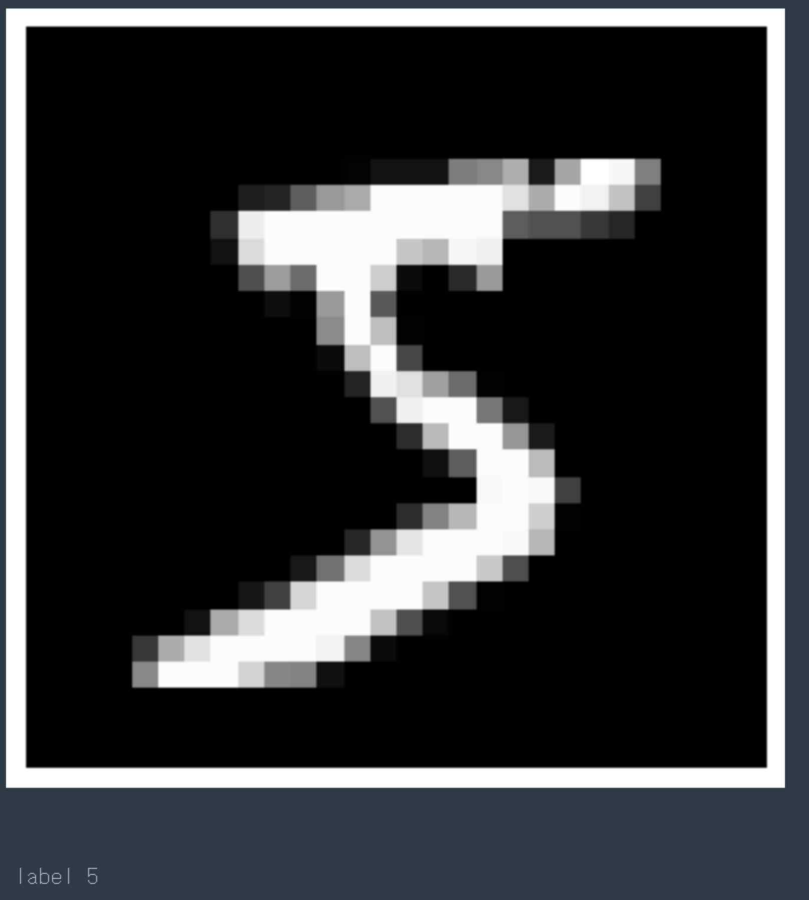
train_set과 test_set의 길이를 확인한다.(각각 60000, 10000)
MNIST의 입력 데이터 형상은 (1, 28, 28)이다.(1채널의 28x28 픽셀 이미지 데이터이다.)
레이블에는 정답 숫자의 인덱스 (0~9)가 들어있다.

```
x, t = train_set[0]
print(type(x), x.shape)
print(t)
```

```
<class 'numpy.ndarray'> (1, 28, 28)
5
```

입력 데이터 시각화

```
x, t = train_set[0]
plt.imshow(x.reshape(28, 28), cmap = 'gray')
plt.axis('off')
plt.show()
print('label', t)
```



```
label 5
```

입력데이터 전처리

(1, 28, 28) 형상인 입력 데이터를 평탄화하여, (784,) 형상으로 변환한다.
데이터 타입을 np.float32로 변환한다.
255.0으로 나누어 값의 범위가 0.0~1.0사이가 되도록 한다.
MNIST(train = True)로 호출하면 위의 과정이 자동으로 수행된다.
dezero/transforms.py에 정의된 클래스를 이용하여 전처리하도록 작성한다.

MNIST 학습하기

이전 단계와 달라진 점은 MNIST 데이터셋 사용과 하이퍼파라미터 값이 변경되었다.
accuracy는 테스트 데이터셋에서 0.8547을 얻었다.(인식 정확도 85.47%)

```
max_epoch = 5
batch_size = 100
hidden_size = 1000

train_set = MNIST(train = True)
test_set = MNIST(train = False)
train_loader = DataLoader(train_set, batch_size)
test_loader = DataLoader(test_set, batch_size, shuffle = False)

model = MLP((hidden_size, 10))
optimizer = optimizers.SGD().setup(model)

for epoch in range(max_epoch):
    sum_loss, sum_acc = 0, 0

    for x, t in train_loader:
        y = model(x)
        loss = F.softmax_cross_entropy(y, t)
        acc = F.accuracy(y, t)
        model.cleargrads()
        loss.backward()
        optimizer.update()

        sum_loss += float(loss.data) * len(t)
        sum_acc += float(acc.data) * len(t)
    print('epoch: {}'.format(epoch + 1))
    print('train lossL {:.4f}, accuracy: {:.4f}'.format(
        sum_loss / len(train_set), sum_acc / len(train_set)))
```

```
epoch: 5
train lossL 0.6336, accuracy: 0.8547
```

모델 개선하기

활성화 함수를 sigmoid에서 ReLU로 변경한다.
최적화 기법(optimizer)을 SGD에서 Adam으로 변경한다.

```
model = MLP((hidden_size, 10), activation = F.relu)# 활성함수를 sigmoid에서 relu로 변경
optimizer = optimizers.Adam().setup(model)# SGD를 Adam으로 변경
```

```
epoch: 5
train lossL 0.0256, accuracy: 0.9927
```

accuracy는 테스트 데이터셋에서 0.9927을 얻었다.(인식 정확도 99.27%)

지금까지의 구현으로 딥러닝 프레임워크라고 부를 수 있게 되었다.