Computer System Lab.

Keras와 Pytorch 라이브러리의 딥러닝 모델 비교

순천향대학교
컴퓨터시스템연구실

이인규

21.11.23

SCH University

목표

- Keras와 Pytorch의 학습 속도 비교
- Keras와 Pytorch의 소스 코드 비교

모델 구조

입력층
- 입력크기: 784 (28 * 28 의 이미지)

은닉층 (1층)
- 노드: 128개
- 활성화 함수 : relu

출력층
- 노드: 10개
- 활성화 함수: softmax

옵션
- 오차함수: CrossEntropy
- 최적화 함수: Adam
- 학습률: 0.01
- 반복횟수(epoch): 20
- 미니배치 크기: 64

학습 데이터 불러오기

In [32]:
```
1  import pandas as pd
2
3  train_data = pd.read_csv('mnist_train.csv')
4  test_data = pd.read_csv('mnist_test.csv')
5
6  train_data.shape, test_data.shape
```

executed in 1.91s, finished 10:03:47 2022-11-23

Out[32]: ((60000, 785), (10000, 785))

픽셀데이터(784) + 라벨(1)

데이터 전처리

◆ 최소-최대 정규화 (Min-Max Normalization)

In [33]:

```
X_train = train_data.drop(['label'], axis=1)
y_train = train_data['label']
X_test = test_data.drop(['label'], axis=1)
y_test = test_data['label']

X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
X_train = X_train / 255
X_test = X_test / 255

X_train.shape, y_train.shape, X_test.shape, y_test.shape
```

executed in 137ms, finished 10:03:47 2022-11-23

Out [33]: ((60000, 784), (60000,), (10000, 784), (10000,))

딥러닝 모델 객체 생성

Pytorch

```
7  class FC_NN(nn.Module):
8      def __init__(self):
9          super(FC_NN, self).__init__()
10         self.input_layer = nn.Linear(784, 128) # input_node, output_node
11         self.hidden_layer = nn.Linear(128, 10) # input_node, output_node
12
13     def forward(self, x):
14         x = self.input_layer(x)
15         x = F.relu(x)
16         output = self.hidden_layer(x)
17
18         return output
```

Tensorflow

```
5  model = Sequential()
6  model.add(Dense(128, input_shape=(784,), activation='relu'))
7  model.add(Dense(10, activation='softmax'))
```

딥러닝 모델 옵션 지정

Pytorch

```
20 torch_model = FC_NN()
21 loss_fn = nn.CrossEntropyLoss() # softmax 포함
22 optimizer = torch.optim.Adam(torch_model.parameters(), lr=0.01)
```

Tensorflow

```
8 model.compile(loss='sparse_categorical_crossentropy', optimizer=optimizers.Adam(0.01), metrics=['accuracy'])
```

Pytorch 미니배치 데이터 생성

Pytorch

```
24 torch_x_train = torch.FloatTensor(X_train.values)
25 torch_y_train = torch.FloatTensor(y_train.values).long()
26 torch_x_test = torch.FloatTensor(X_test.values)
27 torch_y_test = torch.FloatTensor(y_test.values).long()
28
29 torch_train = TensorDataset(torch_x_train, torch_y_train)
30 torch_test = TensorDataset(torch_x_test, torch_y_test)
31
32 torch_train_loader = DataLoader(dataset=torch_train, batch_size = 64, shuffle=True)
33 torch_test_loader = DataLoader(dataset=torch_test, batch_size = 64, shuffle=False)
```

Tensorflow

학습 과정에서 자동으로 나누어준다.

딥러닝 모델 학습

Pytorch

```
torch_model.train()
n_epochs = 20
i = 1

for epoch in range(n_epochs):
    avg_loss = 0
    total_batch = len(torch_train_loader)

    for data, targets in torch_train_loader:
        optimizer.zero_grad()

        x = data.view(-1, 784)
        prediction = torch_model(x)
        loss = loss_fn(prediction, targets)
        loss.backward()
        optimizer.step()
        avg_loss += loss / total_batch

    print(f"Epoch: {epoch+1}, Loss: {avg_loss:.4f}")
print("end")
```

Tensorflow

```
model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=20, batch_size=64)
```

딥러닝 학습 속도, 오차, 정확도 비교

◆ **조건**

1. 동일한 모델
2. 훈련 데이터의 정확도 출력
3. 훈련 데이터의 오차 출력
4. 테스트 데이터의 정확도 출력
5. 미니배치를 나누는 과정 포함
6. 반복횟수(epoch)마다 오차 출력 안함
   (Tensorflow에서 verbose=False)

딥러닝 학습 속도, 오차, 정확도 비교

```python
def Torch_version(batch_size, epoch):
    torch_model = FC_NN()
    loss_fn = nn.CrossEntropyLoss() # softmax 포함
    optimizer = torch.optim.Adam(torch_model.parameters(), lr=0.01)
    torch_train_loader = DataLoader(dataset=torch_train, batch_size = batch_size)
    torch_test_loader = DataLoader(dataset=torch_test, batch_size = batch_size)

    torch_model.train()
    n_epochs = epoch
    i = 1

    for epoch in range(n_epochs):
        avg_loss = 0
        total_batch = len(torch_train_loader)

        for data, targets in torch_train_loader:
            optimizer.zero_grad()

            x = data.view(-1, 784)
            prediction = torch_model(x)
            loss = loss_fn(prediction, targets)
            loss.backward()
            optimizer.step()
            avg_loss += loss / total_batch

    torch_model.eval()

    with torch.no_grad():
        correct_train = 0; correct_test = 0
        for data, targets in torch_train_loader:
            x = data.view(-1, 784)
            prediction = torch_model(x)
            output, predicted = torch.max(prediction, 1)
            correct_train += predicted.eq(targets).sum()
        data_num_train = len(torch_train_loader.dataset)

        correct = 0
        for data, targets in torch_test_loader:
            x = data.view(-1, 784)
            prediction = torch_model(x)
            output, predicted = torch.max(prediction, 1)
            correct_test += predicted.eq(targets).sum()
        data_num_test = len(torch_test_loader.dataset)

    return avg_loss, correct_train / data_num_train, correct_test / data_num_test
```

딥러닝 학습 속도, 오차, 정확도 비교

```
72  def Tensor_version(batch_size, epoch):
73      model = Sequential()
74      model.add(Dense(128, input_shape=(784,), activation='relu'))
75      model.add(Dense(10, activation='softmax'))
76      model.compile(loss='sparse_categorical_crossentropy', optimizer=optimizers.Adam(0.01), metrics=['accuracy'])
77      history = model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=epoch, batch_size=batch_size, verbose=False)
78
79      return history.history['loss'][-1], history.history['accuracy'][-1], history.history['val_accuracy'][-1]
```

딥러닝 학습 속도, 오차, 정확도 비교

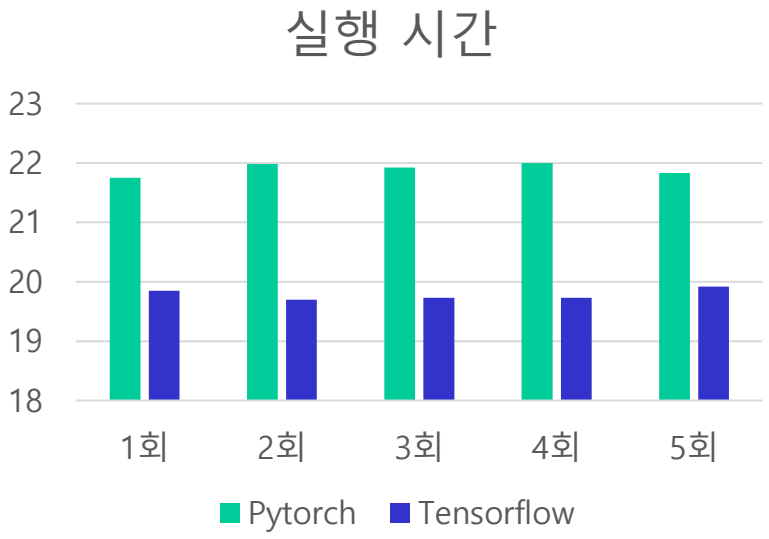
```python
import time

for i in range(1,6):
    start = time.time()
    loss, acc, val_acc = Torch_version(64, 20)
    end = time.time()
    print(f"| PyTorch {i} | time: {end-start:.4f} | loss: {loss:.4f} | acc: {acc:.4f} | val_acc: {val_acc:.4f} |")
```

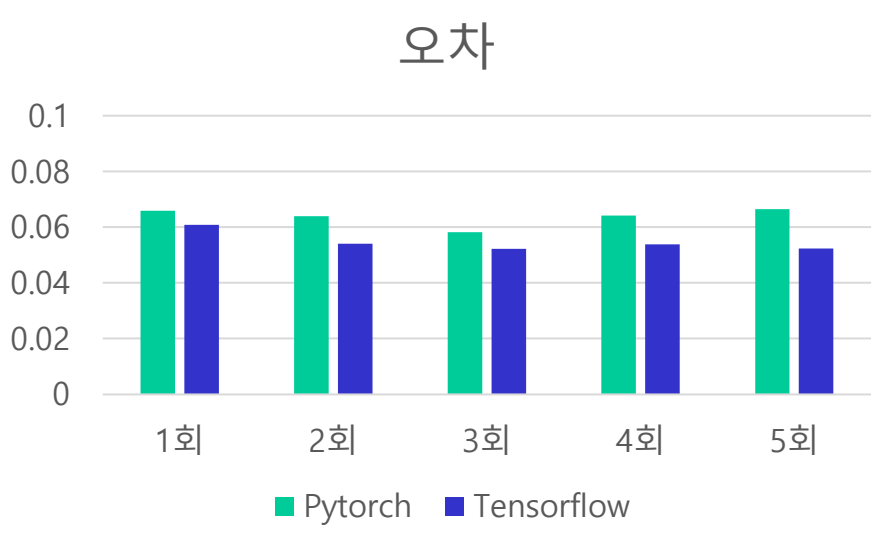
```
| PyTorch 1 | time: 21.7537 | loss: 0.0659 | acc: 0.9883 | val_acc: 0.9699 |
| PyTorch 2 | time: 21.9843 | loss: 0.0639 | acc: 0.9891 | val_acc: 0.9713 |
| PyTorch 3 | time: 21.9285 | loss: 0.0582 | acc: 0.9861 | val_acc: 0.9662 |
| PyTorch 4 | time: 22.0075 | loss: 0.0642 | acc: 0.9845 | val_acc: 0.9680 |
| PyTorch 5 | time: 21.8310 | loss: 0.0665 | acc: 0.9900 | val_acc: 0.9693 |
```

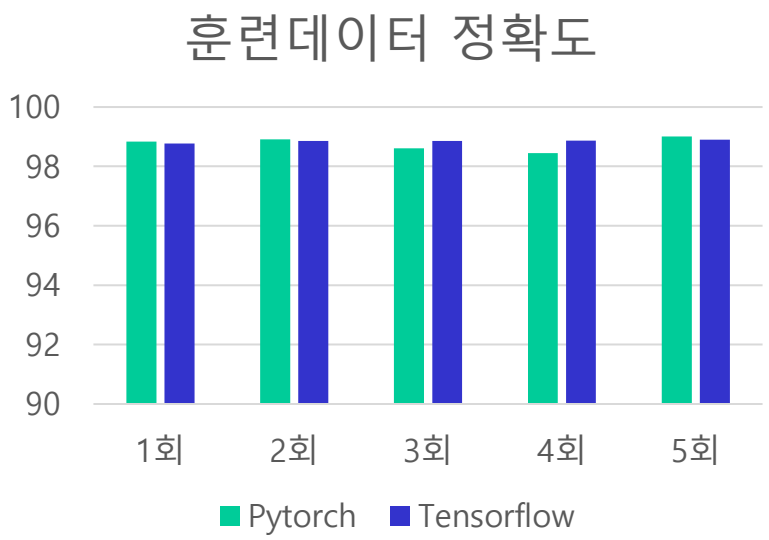
```python
import time

for i in range(1,6):
    start = time.time()
    loss, acc, val_acc = Tensor_version(64, 20)
    end = time.time()
    print(f"| Tenflow {i} | time: {end-start:.4f} | loss: {loss:.4f} | acc: {acc:.4f} | val_acc: {val_acc:.4f} |")
```

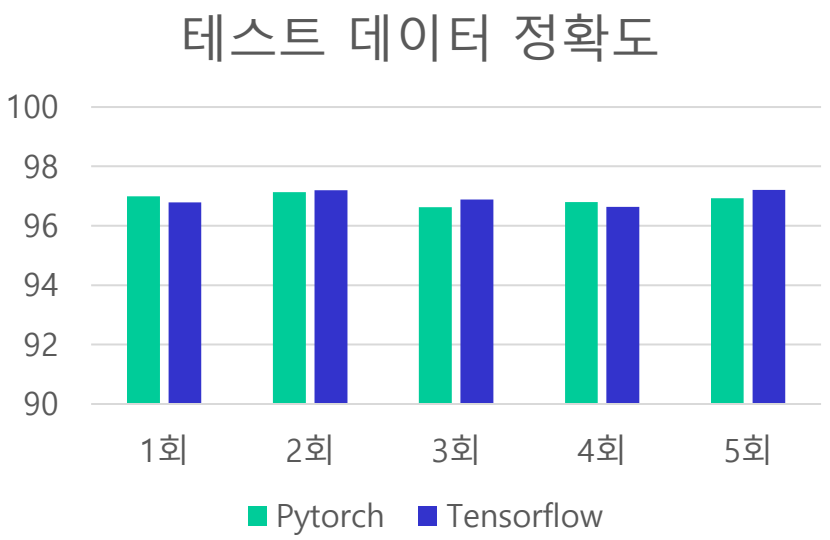
```
| Tenflow 1 | time: 19.8539 | loss: 0.0608 | acc: 0.9877 | val_acc: 0.9679 |
| Tenflow 2 | time: 19.7092 | loss: 0.0540 | acc: 0.9885 | val_acc: 0.9720 |
| Tenflow 3 | time: 19.7319 | loss: 0.0522 | acc: 0.9889 | val_acc: 0.9688 |
| Tenflow 4 | time: 19.7392 | loss: 0.0538 | acc: 0.9887 | val_acc: 0.9664 |
| Tenflow 5 | time: 19.9213 | loss: 0.0523 | acc: 0.9890 | val_acc: 0.9721 |
```

딥러닝 학습 속도, 오차, 정확도 비교









결론

- 소스코드의 길이나 가독성으로는 Tensorflow가 압도적으로 좋은 것 같다.
- Tensorflow에서는 어려운 측정 수단(Metrics)의 변경이 Pytorch에서는 Python레벨에서 변경가능하다.
- 또한 early stopping 등등 다양한 최적화 또한 구현이 쉬울 것 같다.

추가

What's New In Python 3.11

Release:	3.11.0
Date:	November 22, 2022
Editor:	Pablo Galindo Salgado

This article explains the new features in Python 3.11, compared to 3.10.

For full details, see the changelog.

Summary – Release highlights

- Python 3.11 is between 10-60% faster than Python 3.10. On average, we measured a 1.25x speedup on the standard benchmark suite. See Faster CPython for details.

요약 – 릴리스 하이라이트

- Python 3.11은 Python 3.10보다 10-60% 더 빠릅니다. 평균적으로 표준 벤치마크 제품군에서 1.25배의 속도 향상을 측정했습니다. 자세한 내용은 더 빠른 CPython 을 참조하십시오.

개발환경 및 레퍼런스

개발환경

IP: 220.69.209.124

CPU: AMD Ryzen 9 3950x 16-Core Processor * 2

Python: Python 3.6.9

Tensorflow: 2.6.2

Pytorch: 1.10.2

레퍼런스

프로그래밍 기초2: 22-9-Prog2-ML1, 22-9-Prog2-ML2

https://www.kaggle.com/datasets/oddrationale/mnist-in-csv

Question?

Please contact :

이인규
순천향대학교 컴퓨터학부
멀티미디어관 M606

Email : dldlsrb1414@naver.com