딥러닝 스터디

과제 – 와인

20223114 서예은

CONTENTS

1. 개요
2. 신경망 구성
3. 코드 리뷰
4. 결과

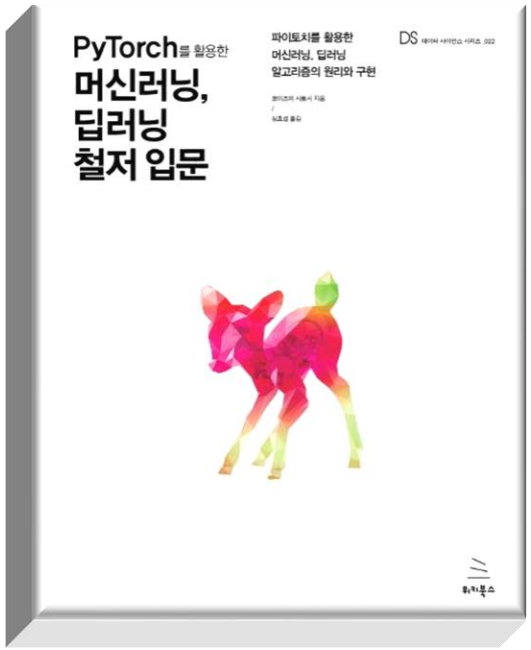
1. 개요

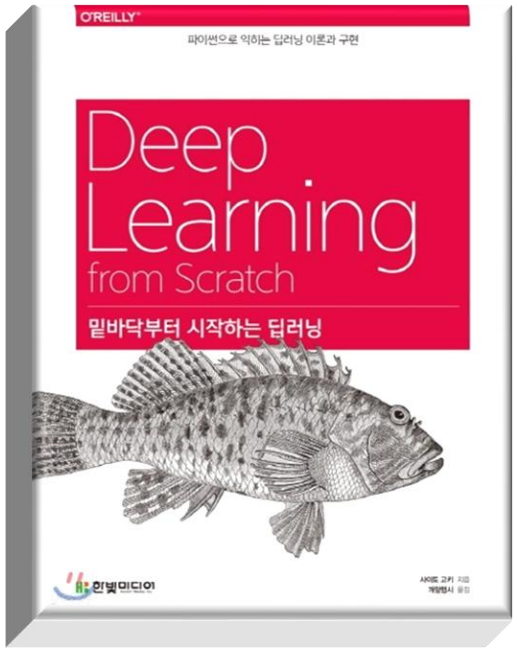
참고 문헌

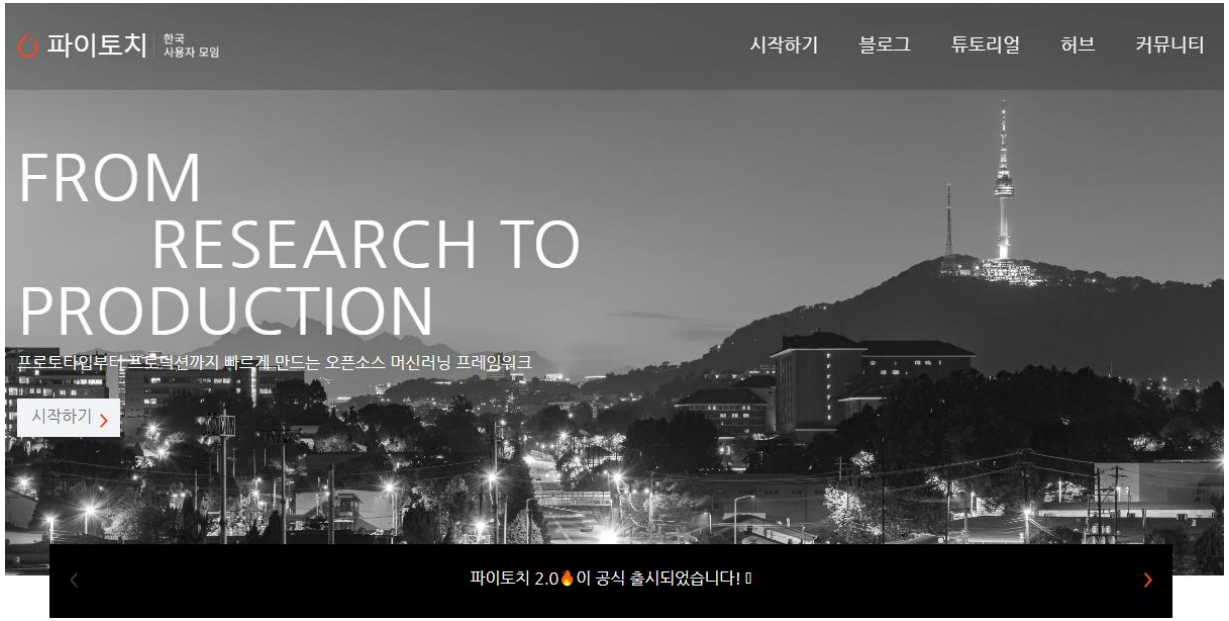
1. PyTorch를 활용한 머신러닝, 딥러닝 철저 입문
2. 밑바닥부터 시작하는 딥러닝

참고 사이트

- 파이토치 공식 사이트







1. 개요

라이브러리

1. Pytorch
   - **신경망 구축**에 사용되는 소프트웨어 기반 오픈소스 러닝 프레임워크
2. Pandas
   - **데이터 처리**와 **분석**을 위한 라이브러리
3. Numpy
   - 대규모 **다차원 배열과 행렬** 연산에 필요한 다양한 함수와 메소드를 제공하는 라이브러리
4. Mataplotlib
   - **데이터 시각화**와 2D 그래프 플롯에 사용되는 라이브러리
5. Sklearn
   - **머신러닝 분석**할 때 유용하게 사용할 수 있는 라이브러리

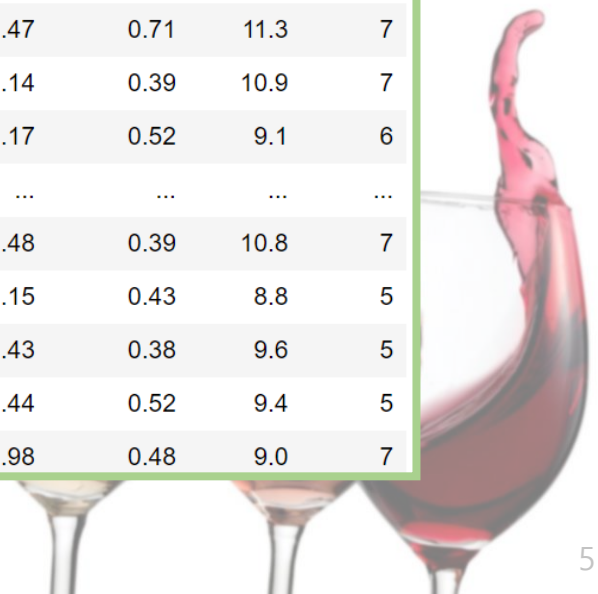
1. 개요

와인 데이터

- 설명 변수 : 레드 와인, 화이트 와인 (0,1)
- 목적 변수 : 총 12 가지의 와인 성분

	label	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	0	4.6	0.52	0.15	2.1	0.054	8.0	65.0	0.99340	3.90	0.56	13.1	4
1	1	6.2	0.25	0.25	1.4	0.030	35.0	105.0	0.99120	3.30	0.44	11.1	7
2	0	6.4	0.57	0.12	2.3	0.120	25.0	36.0	0.99519	3.47	0.71	11.3	7
3	1	7.8	0.30	0.40	1.8	0.028	23.0	122.0	0.99140	3.14	0.39	10.9	7
4	1	7.1	0.25	0.32	10.3	0.041	66.0	272.0	0.99690	3.17	0.52	9.1	6
...	...	...	...	...	...	...	...	...	...	...	...	...	...
1995	1	5.0	0.17	0.56	1.5	0.026	24.0	115.0	0.99060	3.48	0.39	10.8	7
1996	1	6.4	0.21	0.50	11.6	0.042	45.0	153.0	0.99720	3.15	0.43	8.8	5
1997	1	7.3	0.25	0.29	7.5	0.049	38.0	158.0	0.99650	3.43	0.38	9.6	5
1998	0	7.2	0.41	0.30	2.1	0.083	35.0	72.0	0.99700	3.44	0.52	9.4	5
1999	1	7.2	0.23	0.39	14.2	0.058	49.0	192.0	0.99790	2.98	0.48	9.0	7

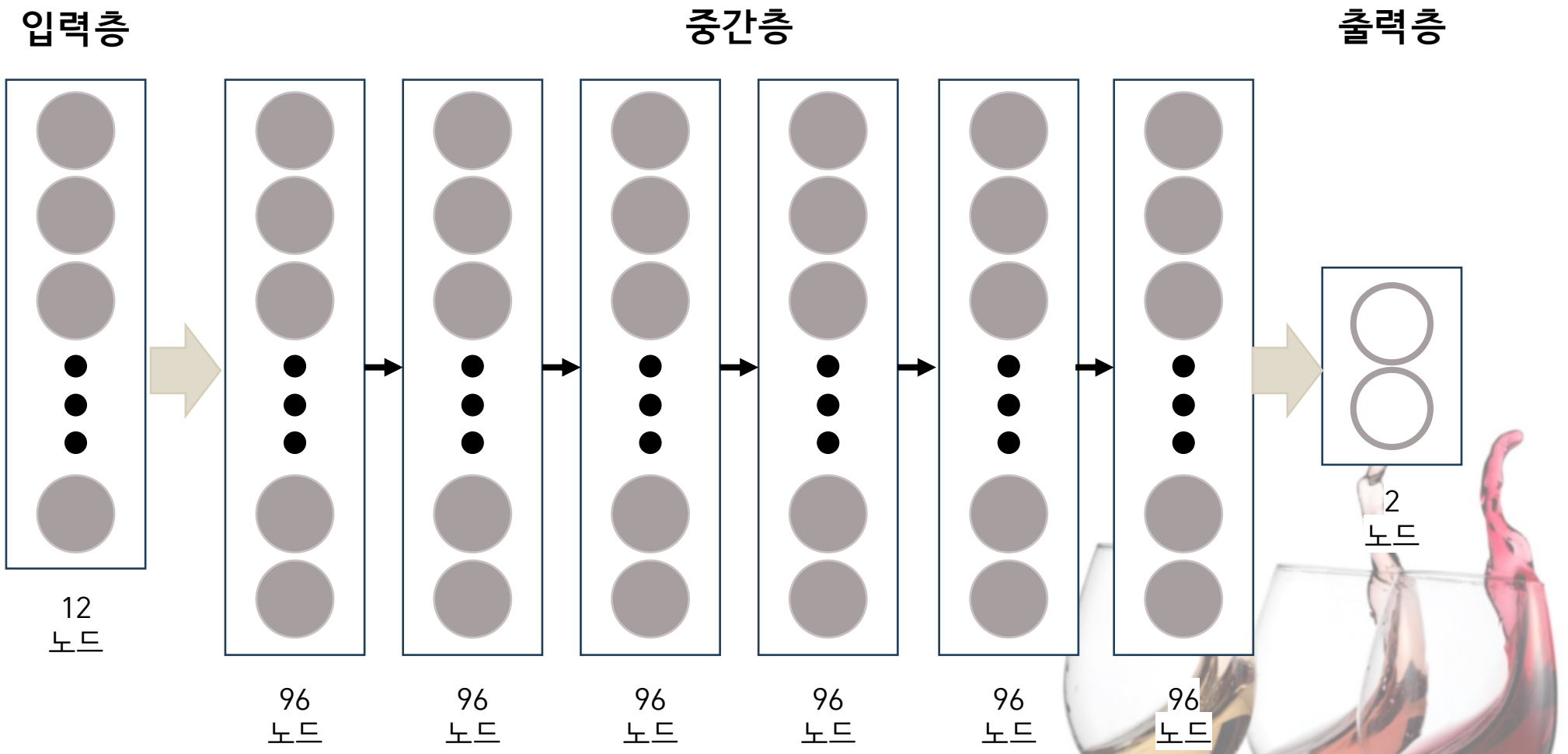
▲ Train Data



2. 신경망 구성

입력
(설명 변수)

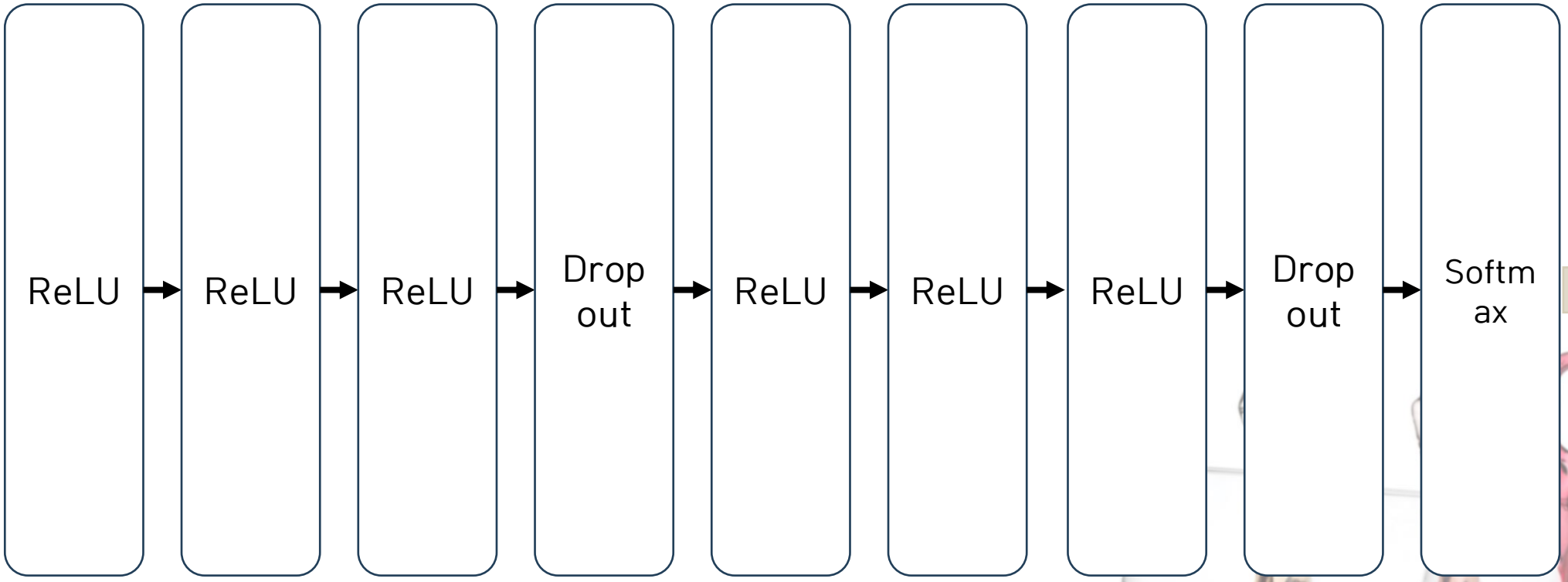
fixed acidity
volatile acidity
citric acid
residual sugar
chlorides
free sulfur dioxide
total sulfur dioxide
density
pH
Sulphates
alcohol
quality



2. 신경망 구성

입력
(설명 변수)

fixed acidity
volatile acidity
citric acid
residual sugar
chlorides
free sulfur dioxide
total sulfur dioxide
density
pH
Sulphates
alcohol
quality



3. 코드 리뷰

```
train_df = pd.read_csv('train.csv')
train_df = pd.DataFrame(train_df)

wine_target =train_df['label']
df_attr = train_df.drop(['label'],axis=1)

wine_data = df_attr[0:2000]
wine_target = wine_target[0:2000]
#Splitting
x_train, x_test, y_train, y_test = train_test_split(wine_data, wine_target,test_size=0.2)

#Numpy
x_train = np.array(x_train).astype(np.float32)
y_train = np.array(y_train).astype(np.longlong)
x_test = np.array(x_test).astype(np.float32)
y_test = np.array(y_test).astype(np.longlong)
#Tensor
x_train = torch.from_numpy(x_train).float()
y_train = torch.from_numpy(y_train).long()
x_test = torch.from_numpy(x_test).float()
y_test = torch.from_numpy(y_test).long()

train = TensorDataset(x_train, y_train)
train_loader = DataLoader(train, batch_size = 16, shuffle=True)
```

훈련 데이터를 가져와 분류

훈련/ 테스트 데이터로 분할

파이토치로 다루기 위해
텐서로 변환(넘파이 -> 텐서)

설명 변수와 목적 변수의 텐서를 합침

3. 코드 리뷰

```python
class Net(nn.Module):

    def __init__(self):
        super(Net,self).__init__()
        self.fc1 = nn.Linear(12,96)
        self.fc2 = nn.Linear(96,96)
        self.fc3 = nn.Linear(96,96)
        self.fc4 = nn.Linear(96,96)
        self.fc5 = nn.Linear(96,96)
        self.fc6 = nn.Linear(96,96)
        self.dropout = nn.Dropout(0.25)
        self.fc10 = nn.Linear(96,2)

    def forward(self, x):
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = F.relu(self.fc3(x))
        x = (self.dropout(x))
        x = F.relu(self.fc4(x))
        x = F.relu(self.fc5(x))
        x = F.relu(self.fc6(x))
        x = (self.dropout(x))
        x = self.fc10(x)
        return F.log_softmax(x, dim=1)
```

6개의 은닉층,
2개의 출력을 가진 출력층 구성

드롭아웃 으로 과적합을 방지

파이토치 함수를 사용하여
활성화 함수로 ReLu 함수,
SoftMax 함수로
클래스 확률 출력

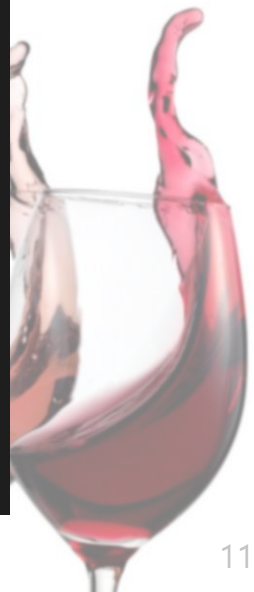
3. 코드 리뷰

```
    def predict(self, x):
        with torch.no_grad():
            x = self.forward(x)
        return x

    def accuracy(self, x, t):
        y = self.predict(x)
        y = torch.argmax(y, dim=1)
        if t.ndim ≠ 1:
            t = torch.argmax(t, dim=1)
        accuracy = torch.sum(y == t).item() / float(x.shape[0])
        return accuracy
```

정확도 측정

3. 코드 리뷰

```
model = Net()
criterion = nn.CrossEntropyLoss() CrossEntropLoss로 오차를 계산
optimizer = torch.optim.Adam(model.parameters(), lr=1e-4, weight_decay=1e-5)
                                  Adam을 적용해 가중치를 최적화

loss_list = []
train_acc_list = []
test_acc_list = []
loss_avg = []

for epoch in range(200):
    total_loss = 0
    for x_train, y_train in train_loader:

        optimizer.zero_grad()경사 초기화
        output = model(x_train)
        loss = criterion(output, y_train) 오차 계산
        loss.backward()
        optimizer.step()가중치 업데이트
        total_loss += loss.item()
```



3. 코드 리뷰

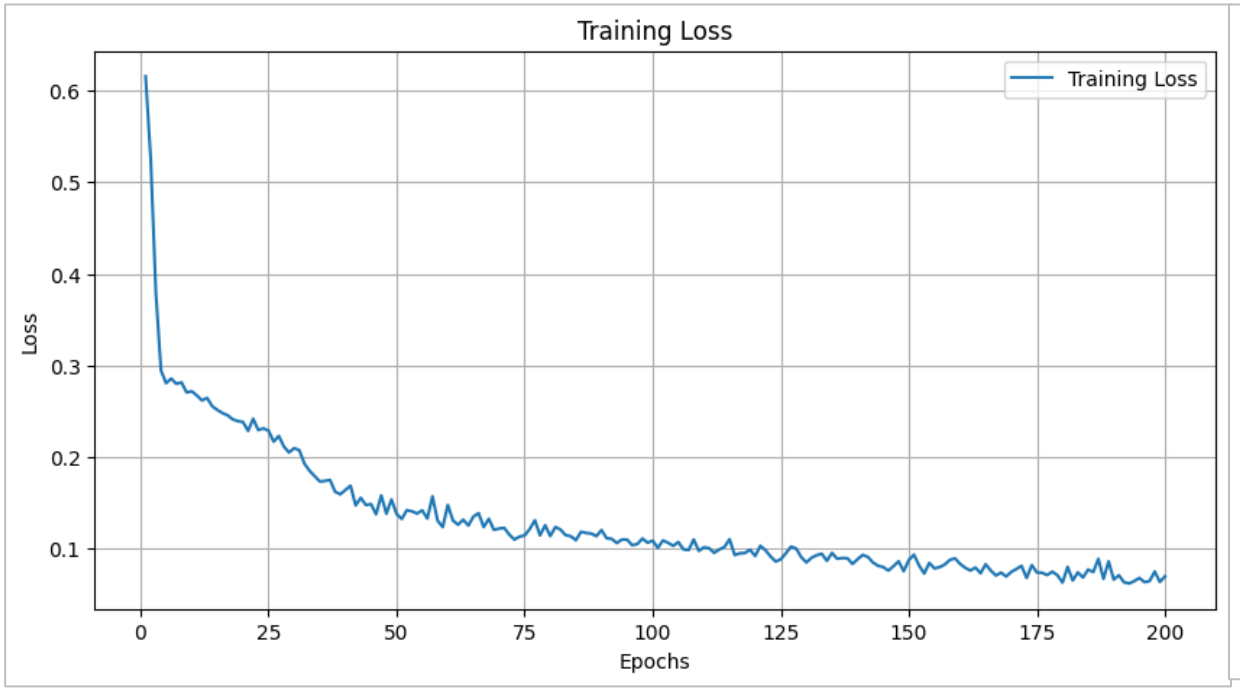
```python
for epoch in range(200):
    for x_train, y_train in train_loader:
        ...
    epoch_loss = total_loss / len(train_loader)
    loss_avg.append(epoch_loss)
    if epoch % 10 ==0:
        print("==========================================")
    print("Epoch: {}, Loss: {:.4f}".format(epoch, total_loss / len(train_loader)))

    train_acc = model.accuracy(x_train, y_train)
    test_acc = model.accuracy(x_test, y_test)
    train_acc_list.append(train_acc)
    test_acc_list.append(test_acc)
    print("train acc:" + str(train_acc) + ", test acc:" + str(test_acc))
```

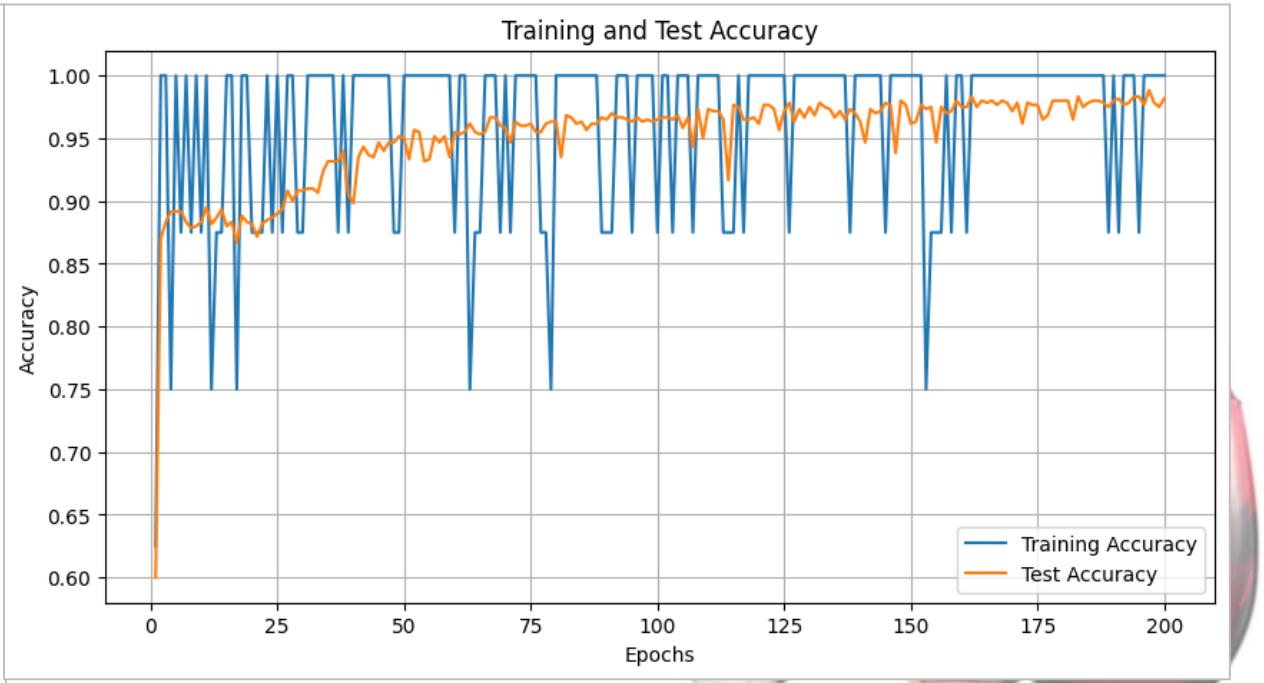
에폭별 평균 손실 계산

에폭별 정확도 계산

4. 결과

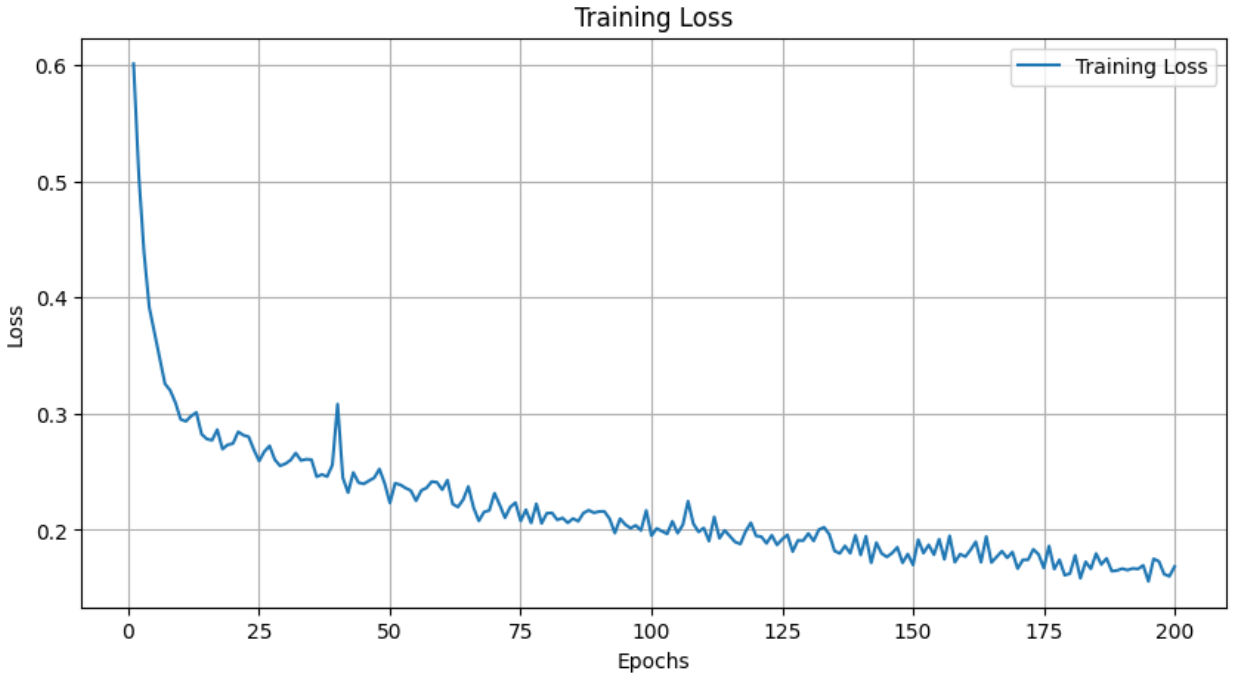
- Accuracy : 약 96.83%
- Adam
- ReLU 3 + Dropout + ReLU 3 + Dropout + Softmax

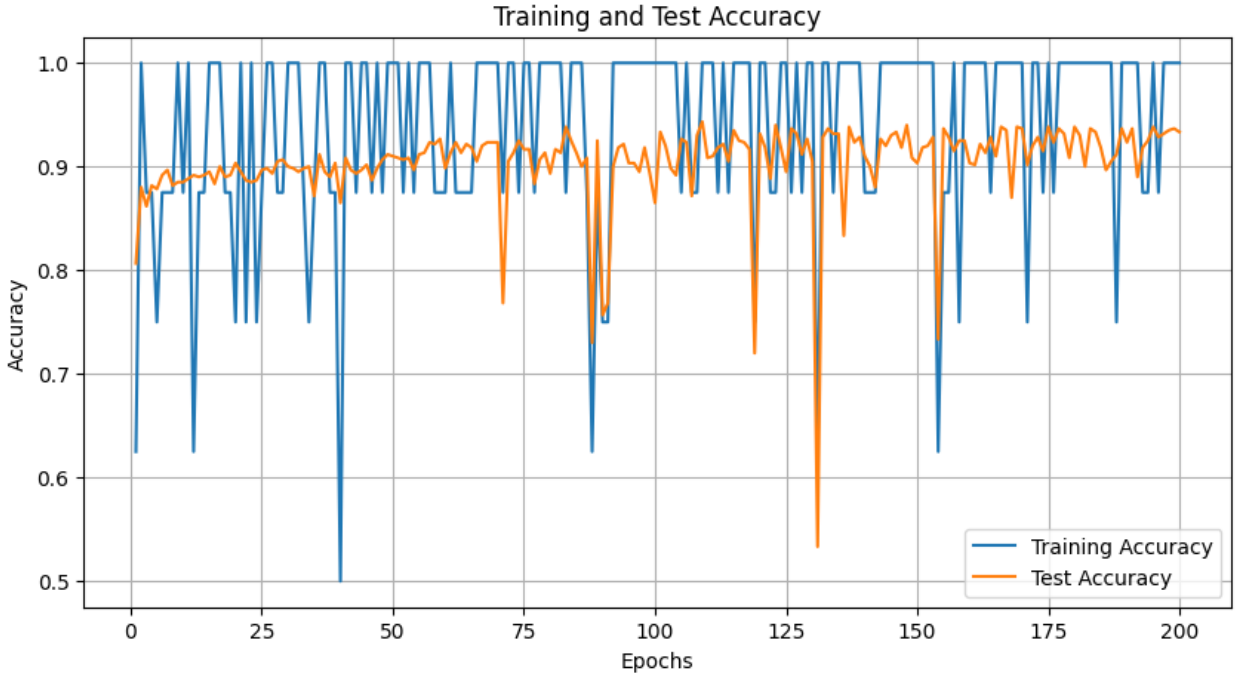




4. 결과

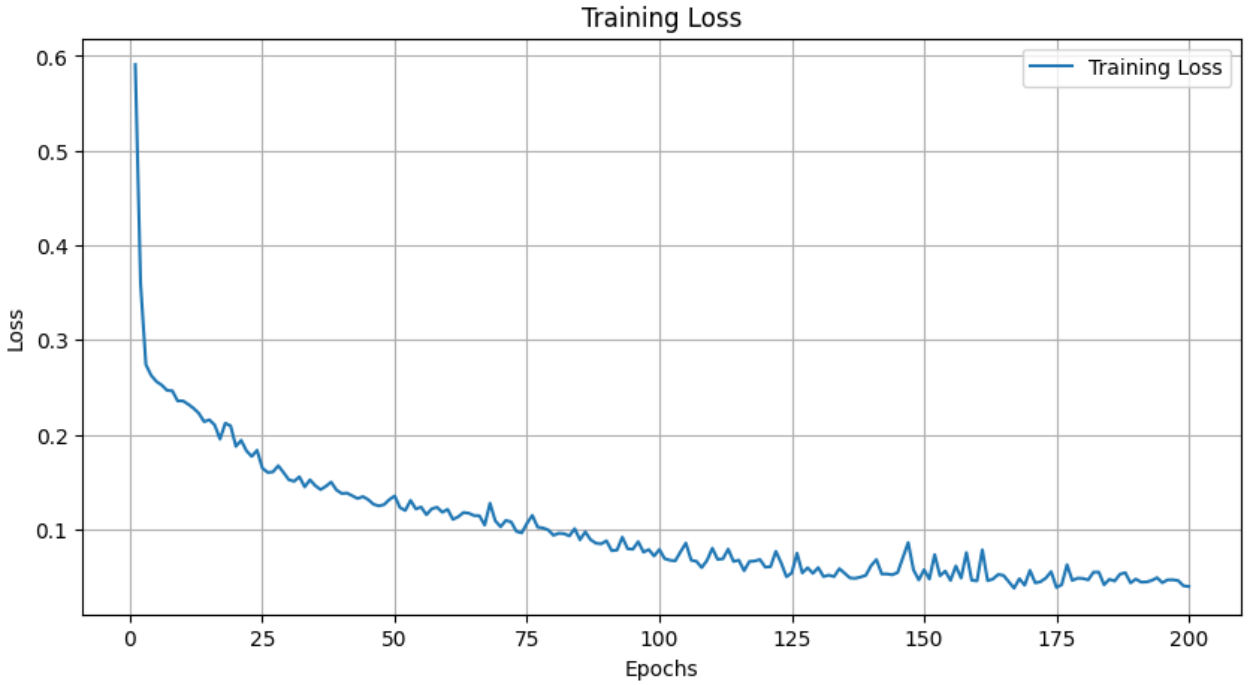
- Accuracy : 약 93.83%
- SGD
- ReLU 3 + Dropout + ReLU 3 + Dropout+ Softmax

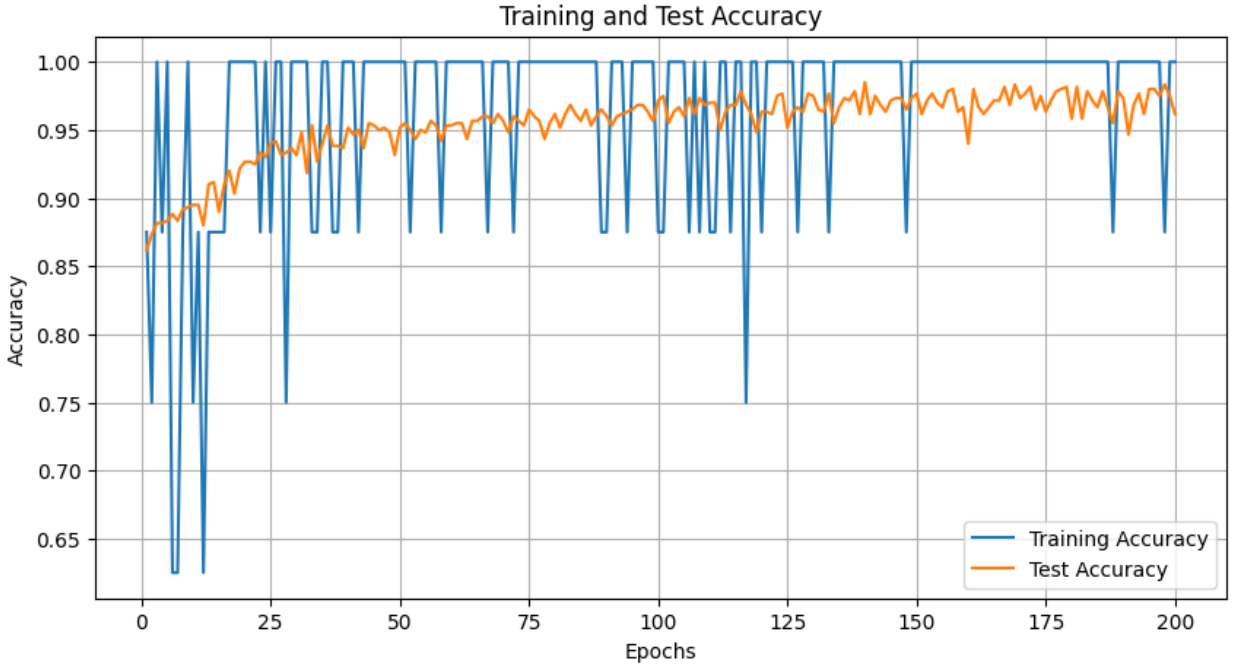




4. 결과

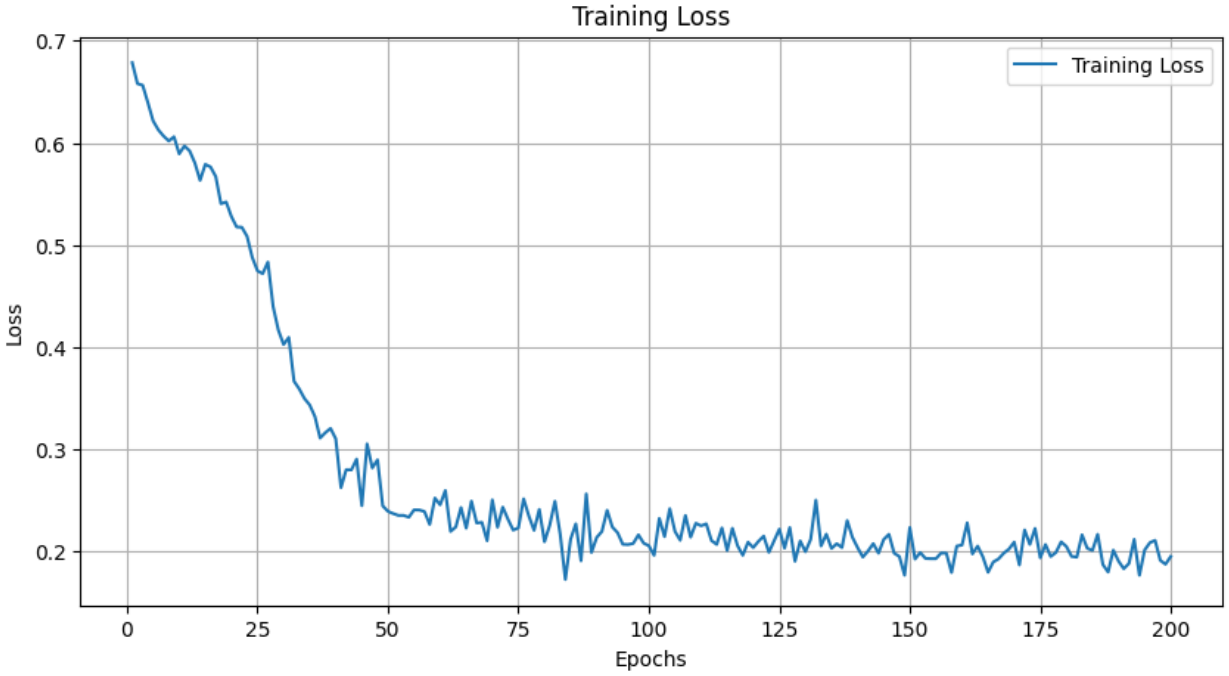
- Accuracy : 약 96.16%
- Adam
- ReLU 6 + Softmax

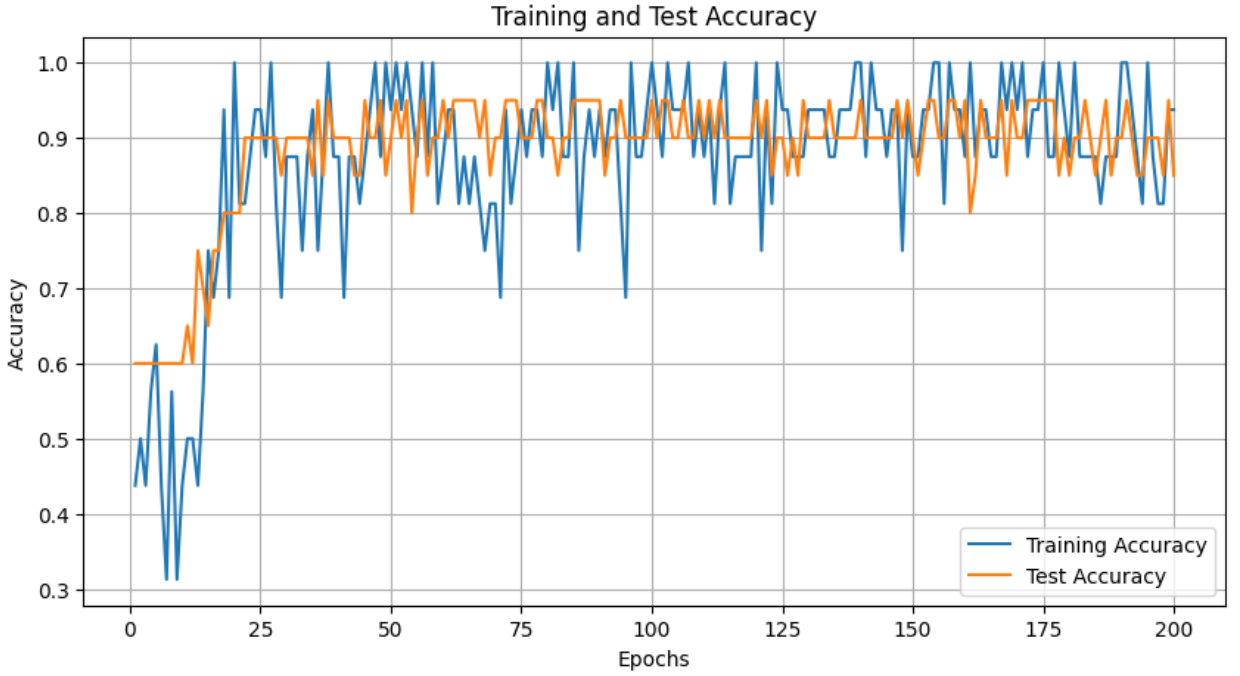




4. 결과 – 추가 데이터

- Accuracy : 약 90%
- Adam
- ReLU 3 + Dropout + ReLU 3 + Softmax





THANK YOU!