

## < 인공지능 Term project PCA Vs. t-SNE >

산업경영공학과  
2017010055 박현일

### 1. 중요코드 설명

#### 1.1 torch model MLP

```
class MLP(nn.Module):
    def __init__(self):
        super().__init__()

        #Set dimensions
        self.in_dim = 28*28
        self.out_dim = 10

        #Set perceptrons
        self.fc1 = nn.Linear(self.in_dim, 512)
        self.fc2 = nn.Linear(512, 256)
        self.fc3 = nn.Linear(256, 128)
        self.fc4 = nn.Linear(128, 64)
        self.fc5 = nn.Linear(64, self.out_dim)

        #Set activation functions
        self.relu = nn.ReLU()
        self.log_softmax = nn.LogSoftmax()

    #Set procedure
    def forward(self, x):
        a1 = self.relu(self.fc1(x.view(-1, self.in_dim)))
        a2 = self.relu(self.fc2(a1))
        a3 = self.relu(self.fc3(a2))
        a4 = self.relu(self.fc4(a3))
        logit = self.fc5(a4)

        return logit

    #Get outputs that passed layers for X
    def z1(self, x):
        return self.fc1(x.view(-1, self.in_dim))

    def a1(self, x):
        return self.relu(self.z1(x))

    def z2(self, x):
        return self.fc2(self.a1(x))

    def a2(self, x):
        return self.relu(self.z2(x))

    def z3(self, x):
        return self.fc3(self.a2(x))

    def a3(self, x):
        return self.relu(self.z3(x))

    def z4(self, x):
        return self.fc4(self.a3(x))

    def a4(self, x):
        return self.relu(self.z4(x))

    def z5(self, x):
        return self.fc5(self.a4(x))

    def a5(self, x):
        return self.log_softmax(self.z5(x), dim=-1)
```

- MLP 클래스를 initialize 할 때 하이퍼 파라미터들을 설정한다.
- MNIST 손글씨 그림의 픽셀은 28\*28 개 이므로 input dimension 은 28\*28, 구별해야하는 숫자는 0~9 이므로 output dimension 은 10 이다.
- 4 개의 Hidden layer 를 가지는 모델을 구성하였고 각

layer 에서 출력값의 PCA, t-SNE 결과를 확인하기위해 layer 구성을 하드코딩 하였다. 이때 각 레이어의 퍼셉트론 수는 [512(z1, a1), 256(z2, a2), 128(z3, a3), 64(z4, a4)]개 이다. Output 레이어(z5, a5)는 0~9 까지 숫자를 예측하는 10 개의 퍼셉트론으로 구성되어있다.

Input	H1	H2	H3	H4	Output
(784)	(512)	(256)	(128)	(64)	(10)

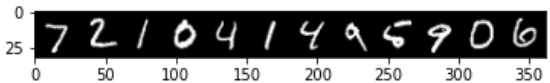
- 각 레이어에서 사용할 활성화 함수 ReLU, LogSoftmax 를 설정하였다.
- 각 레이어가 순서에 맞게 학습할 수 있도록 forward 함수를 구성하였다.
- 각 레이어의 출력값과 활성화 함수 출력값의 PCA, t-SNE 비교를 위해 출력값을 반환하는 함수를 구현하였다.

#### 1.2 MNIST 이미지 출력

```
def imshow(img):
    npimg = img.numpy()
    plt.imshow(np.transpose(npimg, (1, 2, 0)))
    plt.show()

dataiter = iter(test_loader)
images, labels = dataiter.next()

imshow(tv.utils.make_grid(images, nrow=batch_size))
```



- test\_loader 에서 iter 함수를 통해 MNIST 데이터를 batch\_size 만큼 호출하여 출력한다.

#### 1.3 MNIST 이미지 PCA

```
pca = PCA(n_components=2)
pca_result = pca.fit_transform(df[feat_cols].values)
df['pca-one'] = pca_result[:, 0]
df['pca-two'] = pca_result[:, 1]
```

- PCA 클래스를 호출할 때 이미지의 차원을 2 차원으로 축소하기 위해 n\_components=2 를 인자로 넘긴다.
- pca.fit\_transform 함수에 df[feat\_cols].values 인자를 넘겨 PCA 를 수행하고 결과를 데이터프레임에 저장한다.
- df[feat\_cols].values 는 이미지의 픽셀을 (784 X 1) 벡터로 변환한 데이터들의 집합이다.
- PCA 는 전체 이미지에 대해 수행한다.

#### 1.4 MNIST 이미지 t-SNE

```

N = 10000
df_subset = df.loc[rndperm[:N], :].copy()
data_subset = df_subset[feat_cols].values

tsne = TSNE(n_components=2, perplexity=40, n_iter=300)
tsne_results = tsne.fit_transform(data_subset)
df_subset['tsne-2d-one'] = tsne_results[:, 0]
df_subset['tsne-2d-two'] = tsne_results[:, 1]

```

- t-SNE 를 확인하기위한 샘플 10,000 개를 데이터셋에서 랜덤 추출하여 df\_subset 과 data\_subset 을 만든다. 이 때 rndperm 은 데이터셋의 전체 레코드 수 크기를 가지는 랜덤 정수 numpy array 이다.
- PCA 와 마찬가지로 이미지의 차원을 2 차원으로 축소하기 위해 n\_components=2 인자를 사용한다.
- tsne.fit\_transform 함수에 data\_subset 인자를 넘겨 t-SNE 를 수행하고 결과를 데이터프레임에 저장한다.

## 1.5 Layers output PCA

```

z1 = model.z1(torch.from_numpy(df[feat_cols].values).float())
z1 = z1.detach().numpy()

```

```

pca_z1 = PCA(n_components=2)
pca_z1_result = pca_z1.fit_transform(z1)
df['pca-z1-one'] = pca_z1_result[:, 0]
df['pca-z1-two'] = pca_z1_result[:, 1]

```

- 각 레이어를 통과한 데이터 output 의 PCA 결과를 확인하기 위해 model 의 각 함수(ex z1, a2)를 호출하고 결과값을 변수에 저장한다. 이 때 모델 내부의 레이어를 통과하기 위해선 torch tensor 로 변환해야 하기 때문에 torch.from\_numpy 함수를 이용한다.
- PCA 를 확인하기 전 결과값 tensor 를 numpy array 로 변환 한다.
- 나머지 과정은 MNIST 이미지 PCA 와 동일하다.

## 1.6 Layers output t-SNE

```

df_z1_subset = pd.concat([pd.DataFrame(z1[rndperm[:N], :],
                                     index=rndperm[:N]),
                        df.loc[rndperm[:N], :]['y']],
                        axis=1)

data_z1_subset = z1[rndperm[:N], :]

```

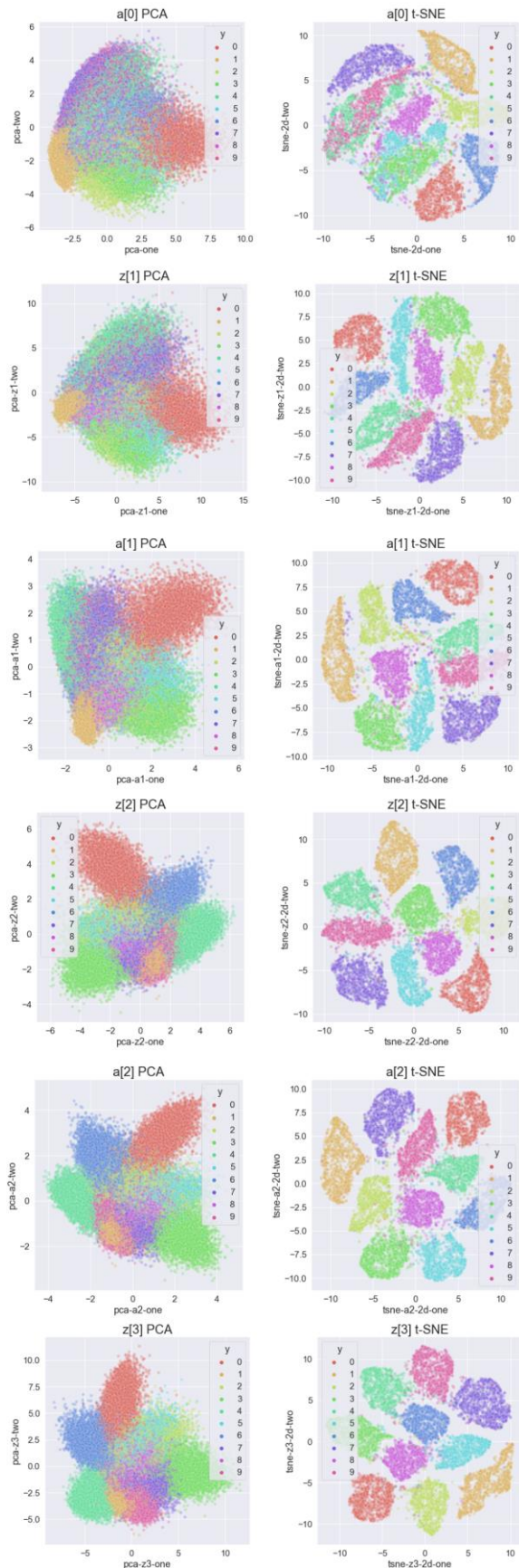
```

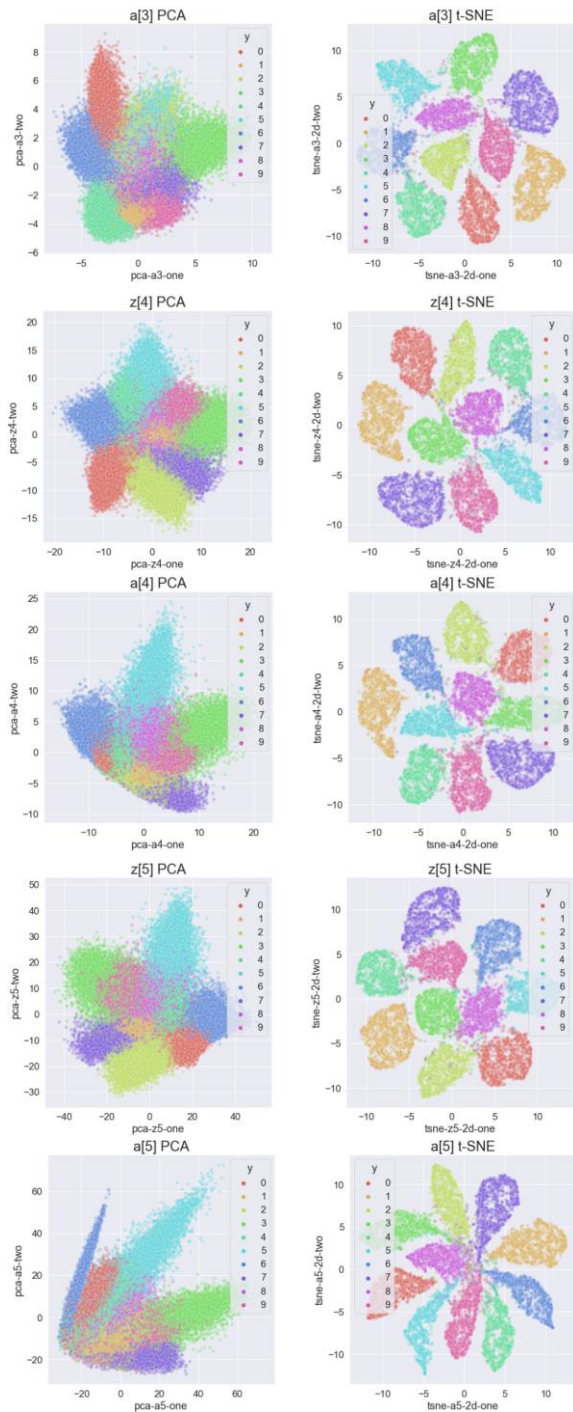
tsne_z1 = TSNE(n_components=2, perplexity=40, n_iter=300)
tsne_z1_results = tsne_z1.fit_transform(data_z1_subset)
df_z1_subset['tsne-z1-2d-one'] = tsne_z1_results[:, 0]
df_z1_subset['tsne-z1-2d-two'] = tsne_z1_results[:, 1]

```

- 위 각 레이어를 통과한 데이터 output 중 rndperm array 를 이용하여 샘플 10,000 개를 추출하여 data\_subset array 를 만들고 원본 데이터셋에서 해당 샘플에 맞는 라벨 값과 합쳐 df\_subset 데이터프레임을 만든다.
- tsne.fit\_transform 함수에 data\_subset 인자를 넘겨 t-SNE 를 수행하고 결과를 데이터프레임에 저장한다.

## 2. 결과 출력





- 모델의 레이어 순서인  $a[0] \rightarrow z[1] \rightarrow a[1] \rightarrow z[2] \rightarrow a[2] \rightarrow z[3] \rightarrow a[3] \rightarrow z[4] \rightarrow a[4] \rightarrow z[5] \rightarrow a[5]$  순으로 출력하였다.

## 2.1 PCA

- MNIST 이미지( $a[0]$ )의 PCA 결과를 보면 일부분을 제외하면 서로 뒤엉켜 그룹의 형태를 확인하기 어렵다.

- 이후 첫번째 hidden 레이어의 활성화 함수( $a[1]$ )를 통과할 때까지 이 현상이 지속되다가 두번째 hidden 레이어를 통과하면서 그룹의 형태가 조금씩 갖춰지기 시작한다.

- 네번째 hidden 레이어 활성화 함수를 통과하기 전( $z[4]$ ) PCA 결과가 가장 깔끔하게 출력되고 조금씩 선형의 모습이 보이기도 한다.

- 마지막 output 레이어의 활성화 함수( $a[5]$ )를 통과했을 때의 PCA 결과가 가장 선형적으로 보이는데 이는 output 레이어의 활성화 함수가 Log\_Softmax 함수이기 때문으로 추측해볼 수 있다.

## 2.2 t-SNE

- MNIST 이미지( $a[0]$ )의 t-SNE 결과를 보면 5 개의 숫자를 제외한 나머지 5 개의 숫자가 서로 뒤엉켜 있는 모습을 확인할 수 있다.

- 이후 진행되는 레이어에서는 이상치를 제외하면 서로 뒤엉키는 현상 없이 잘 분류되고 있는 모습을 보인다.

- 마지막 output 레이어의 활성화 함수( $a[5]$ )를 통과했을 때의 t-SNE 결과가 가장 선형적으로 보이는데 이는 PCA 결과와 마찬가지로 output 레이어의 활성화 함수가 Log\_Softmax 함수이기 때문으로 추측해볼 수 있다.